UNIVERSITÄT BONN

Inaugural-Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Landwirtschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn
Institut für Geodäsie und Geoinformation

Efficient Semantic Scene Understanding for Mobile Robots

von

Andres Leonardo Milioto

aus
Macerata, Italien

Bonn 2021

Referent:
Prof. Dr. Cyrill Stachniss, University of Bonn, Germany

1. Korreferent:
Prof. Dr. Chris McCool, University of Bonn, Germany

2. Korreferent:
Prof. Dr. Abhinav Valada, University of Freiburg, Germany

Tag der mündlichen Prüfung: 27.01.2021

Erscheinungsjahr: 2021

Angefertigt mit Genehmigung der Landwirtschaftlichen Fakultät der Universität Bonn

Zusammenfassung

In den letzten Jahren haben Roboter langsam ihren Weg in unseren Alltag gefunden. Angefangen von Staubsauger-Robotern, die heute schon hinter uns aufräumen, bis hin zur Vision von Flotten von Robo-Taxis und autonom fahrenden Fahrzeugen, sind all jene Roboter dafür geschaffen, im Einklang mit und in einer für uns Menschen gestalteten Umgebung zu operieren. Im Gegensatz zu den herkömmlichen und in der Industrie gebräuchlichen Robotern, die in einer Welt operieren, welche extra für sie konzipiert wurde, benötigen flexible, *mobile Roboter* ein genaues Verständnis ihrer Umwelt um sicher und zuverlässig operieren zu können. Wir bezeichnen diese Art von Wissen über die Umgebung als *semantisches Szenenverständnis*. Dieses Verständnis dient auf der untersten Ebene der Interpretation der rohen Sensordaten und liefert für andere Aufgaben nützliche Informationen über den Zustand der Umgebung. Zu diesen Aufgaben gehören u.a. das Ausweichen vor Hindernissen, die Lokalisierung des Roboters in der Welt, die Kartierung einer unbekannten Umgebung, die Planung von Trajektorien und die Manipulation von Objekten.

Der Fokus dieser Arbeit liegt auf der Schätzung eines semantischen Szenenverständnis für mobile Roboter. Da diese Roboter für ihre Mobilität in der Regel eine batteriebetriebene Energieversorgung benötigen, müssen die verwendeten Algorithmen laufzeittechnisch, sowie energetisch effizient sein. Effizient bedeutet, dass im Ansatz alle zur Verfügung stehenden Informationen schnell genug genutzt werden können, um in Echtzeit zu operieren und das bei sowohl leistungs- als auch resourcentechnisch limitierten Computern. Wir nähern uns diesem Ziel auf drei verschiedenen Wegen. Erstens verwenden wir bei allen in dieser Arbeit vorgestellten Algorithmen Hintergrundwissen über die zu lösende Aufgabe, um unsere Algorithmen schneller auszuführen und gleichzeitig genauere Ergebnisse zu erhalten. Zweitens nutzen unsere Ansätze die spezifischen Besonderheiten des jeweiligen Sensors, für eine effizientere Verarbeitung aus. Drittens stellen wir eine Software-Infrastruktur vor, um die genannten Ansätze zum Szenenverständnis auf realen Robotern umzusetzen, wobei kommerziell erhältliche Hardware-Beschleuniger für die Aufgabe genutzt werden, um eine gute Skalierbarkeit zu ermöglichen. Aus diesem Grund ist jede in dieser Arbeit vorgestellte Methode in der Lage, schneller als die Frequenz des Sensors zu operieren, sowohl beim Einsatz von Kameras als auch von Lasersensoren.

Alle Teile dieser Arbeit wurden auf internationalen Konferenzen oder als Zeitschriftenartikel veröffentlicht, die einem Peer-Review-Verfahren unterzogen wurden. Darüber hinaus führte diese Arbeit zur Veröffentlichung eines groß angelegten Datensatzes und eines öffentlich nutzbaren Benchmarks, um ihre Ansätze zum semantischen Szenenverständnis zu entwickeln, auszutauschen und zu vergleichen. Darüber hinaus wurden vier Open-Source-Bibliotheken für unterschiedliche Sensormodalitäten veröffentlicht.

Abstract

Over the last few years, robots have been slowly making their way into our everyday lives. From robotic vacuum cleaners picking up after us already working in our homes, to the fleets of robo-taxis and self-driving vehicles lurking on the horizon, all of these robots are designed to operate in conjunction with, and in an environment designed for us, humans. This means that unlike traditional robots working in industrial settings where the world is designed around them, these *mobile robots* need to acquire an accurate understanding of the surroundings in order to operate safely, and reliably. We call this type of knowledge about the surroundings of the robot *semantic scene understanding*. This understanding serves as the first layer of interpretation of the robot's raw sensor data and provides other tasks with useful and complete information about the status of the surroundings. These tasks include the avoidance of obstacles, the localization of the robot in the world, the mapping of an unknown environment for later use, the planning of trajectories, and the manipulation of objects in the scene, among others.

In this thesis, we focus on semantic scene understanding for mobile robots. As their mobility usually requires these robots to be powered by batteries, the key characteristics they require from perception algorithms are to be computationally, as well as energy *efficient*. Efficient means that the approach can exploit all the information available to it to run fast enough for the robot's online operation, both in power- as well as compute-constrained embedded computers. We approach this goal through three different avenues. First, in all of the algorithms presented in this thesis, we exploit background knowledge about the task we are trying to solve to make our algorithms fast to execute and at the same time, more accurate. Second, we instruct the approaches to exploit peculiarities of the particular sensor used in each application in order to make the processing more efficient. Finally, we present a software infrastructure that serves as an example of how to implement said scene understanding approaches on real robots, exploiting commercially available hardware accelerators for the task, and allowing for scalability. Because of this, every method presented in this thesis is capable of running faster than the frame rate of the sensor, both when using cameras or laser sensors.

All parts of this thesis have been published in proceedings of international conferences or as journal articles, undergoing a thorough peer-reviewing process. Furthermore, the work presented in this thesis resulted in the publication of a large-scale dataset and benchmark for the community to develop, share, and compare their semantic scene understanding approaches, as well as four open-source libraries for this task, using multiple sensor modalities.

Acknowledgements

WHAT a wild ride this has been. When I started my bachelor's degree in Electrical Engineering in my beautiful hometown of Rosario, Argentina, I never imagined that 10 years later I would be sitting all the way across the world, in Bonn, Germany putting the finishing touches to my doctoral thesis, and having traveled the world in the process. Even though I take full responsibility for the thousands of hours of hard work it has taken to get to this place, there have been many people who have been absolutely instrumental to my getting here, and who I would like to thank right now.

First and foremost, I would like to thank Cyrill Stachniss for being a mentor, through the good times and the bad times, both professionally and personally. I also want to thank him for being relentless in your constructive criticism of my presentation skills, which has been absolutely key in my improvement as a science communicator. As I look back on the person I was three years ago I owe a lot of the positive changes to this guidance.

I would also like to thank Dario Rocha, Mario Munich, Orjeta Taka, Vittorio Ziparo, and Santiago Goldaraz, for guiding me in the process of transitioning out of the safety-cushioned university environment an into the real way products, as well as personal relationships, are developed in the wild. Your support and advice during my time in Rosario, and at iRobot is stored in a very special place in my heart. Also from my time at iRobot I would like to thank Pablo Speciale, Pablo Pilotti, Franco Pestarini, Mauricio Gonzalez Genta, and Brian Lovera, for their friendship and advice during our shared time in Los Angeles and beyond.

I also want to thank my laboratory colleagues with whom I shared fun lunches, hard, long, deadline nights, and wonderful trips. Thank you, Lorenzo Nardi, Igor Bogoslavskyi, Olga Vysotska, Nived Chebrolu, and Xieyuanli Chen for being the best colleagues and friends I could have asked for. I also want to thank Thomas Laebe and Philipp Lottes for their incredible support in my technical endeavors, and for taking the time to listen to me rant about hardware efficiency. Furthermore, I would like to thank Emanuele Palazzolo, Laura Zabawa, Peter Regier, Xieyuanli Chen, Susanne Wenzel, and Ribana Roscher, for collaborating with me in our multiple shared projects, which resulted in the works presented

in Chapter 10. I also thank Birgit Klein, who has been a great support, not only in our work, but also emotionally, and mainly to navigate life as a foreigner in Germany. Finally, I have a very special place reserved to thank Jens Behley, for being the best office mate, for his invaluable advice, for working tirelessly alongside me on multiple occasions, and for his friendship. I also have to thank him for helping me become a better manager, by co-supervising several master's theses with me, including that of Louis Wiesmann, who has quickly become a dear friend, ally, and colleague as well.

In my personal life, I would like to thank Ignacio "Nacho" Vizzo and Mailen Raposo for drinking mate [122] with me and sharing their wisdom when life sometimes got too hard to handle, and for letting me live in their house in the midst of a global pandemic. I also thank Claus Smitt and Evelin Battocchio for having quickly become part of my family.

I also need to thank my family for supporting -as well as tolerating- me through this rocky ride. My sister Giuliana, who is one of my role models, and who has always been, and will always be, right next beside me, as a loyal partner. My uncles and aunts who always remind me there is a place for me in a place I can call home. My grandmothers Norma and Graciela, who against all odds got on a plane to Europe twice to visit me and support me. My grandfather Rafa, who is no longer with us, and who always believed in me. My parents, whose constant support and relentless effort to keep our family afloat in an ever-changing Argentina have got me here, and made me the resilient man I have become. Their love for me and my sister, as well as for one-another always inspired me and served as a wonderful example of what to strive for in life.

And finally, I would like to thank my present and future: Feli, thank you for the hundreds of hours on airplanes after tiring night-shifts, for making the impossible happen by being there with and for me, for tolerating my ups and downs, for daring to dream this crazy dream with me, and for making me a better man.

The works on this thesis have been partly supported by the EC under contract number H2020-ICT-644227-FLOURISH, by the DFG under the grant number FOR 1505: Mapping on Demand, by NVIDIA Corporation, and by the DFG under Germany's Excellence Strategy, EXC-2070 – 390732324 (PhenoRob).

Contents

Chapter 1

Introduction

Robots have been a part of modern society for quite a long time. The first patent for a manufacturing robot was filed over 65 years ago, in 1954, by George Devol [73]. This patent was filed for a robot that could transfer objects between two points in the environment within a distance of less than 3m, controlled by paper tape with hole-punches in it to control levers through solenoids. Since the 1980's, and following Takeo Kanade's invention of the direct-drive arm, which put motors straight into the joints of the arm making them faster and more accurate, robots have quickly and increasingly become a key part of industrial production requiring highly repetitive and precise tasks. So why aren't robots ubiquitous in our every day lives already?

The answer to this question is what motivates every single approach presented in this thesis. To operate safely in the environment, a robot requires an accurate understanding of its surroundings, which allows it to plan paths and actions, and move safely and efficiently without colliding with obstacles, in order to fulfill its purpose. In the case of the aforementioned industrial setting, the knowledge of the environment is given by the programmer by designing the environment to fit the robot. This does not mean that the world around the robot is static, but it means, for example, that the state, and therefore the layout of the world can be extracted directly from the current position in a state machine. Figure 1.1 shows an industrial robot working in a welding and stacking application. Here, the environment is designed around the robot, which is put into a cage, and given pre-defined paths to operate at each time-step, which assumes a static world. This static world assumption is enforced by both, the physical barrier, as well as laser barriers which stop the robot on its tracks when triggered. Furthermore, Figure 1.1 also shows a robot vacuum cleaner moving in an environment that was designed for humans. This does not only mean that the robot operates in a different environment, but also that the robot is *mobile*, which is a key difference between both.

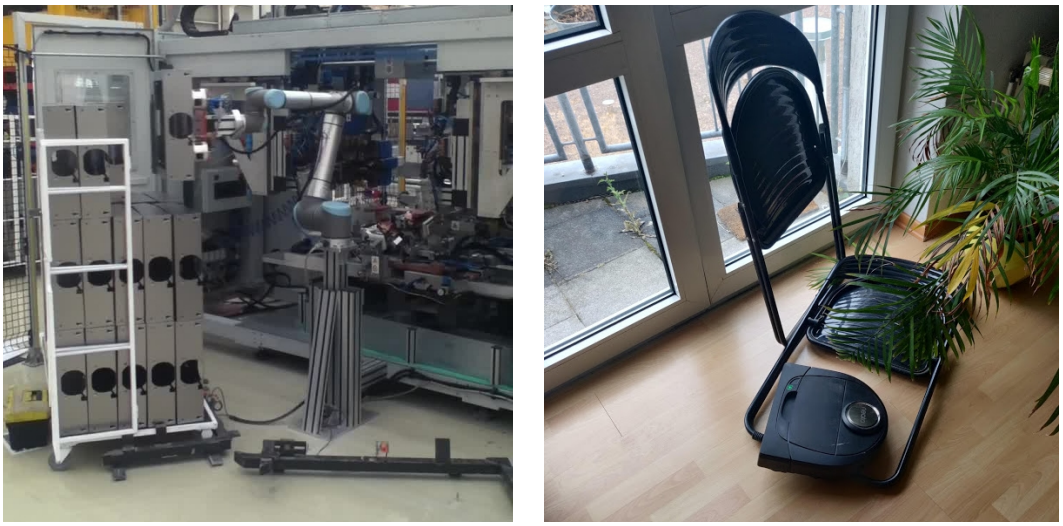

Figure 1.1: Mobile robots operating alongside humans in unstructured environments have wildly different perceptual needs from the successful industrial robots we have been using for decades. Left: Universal Robots UR-10 robot operating within an industrial welding application. Here, the environment is made artificially static while the robot is moving by surrounding it with a cage and a slew of laser safety barriers that trigger an emergency stop. Right: robot vacuum cleaner in a human inhabitable environment. Fencing a vacuum cleaner with laser emergency stops would not only be unfeasibly expensive but also undesired since the purpose of robots working in our homes is convenience. In the depicted scene, and due to a miscalculation in its measurement of the world around it, the vacuum cleaner knocked over a chair and got stuck over it. These are the types of scenarios we aim to avoid, since these robots are supposed to operate when we are not home, and this type of behavior is potentially very dangerous.

Mobile robots that need to operate in an environment designed for humans, such as the inside of our homes or city roads, require a form of semantic scene understanding about the environment through their slew of different sensors. These are sensors such as bumpers switches, odometers, inertial measuring units, and more relevantly to this thesis, contactless sensors such as radar, cameras, and light detection and ranging (LiDAR). In the 1960's, and parallelly to the industrial robotics movement, the problem of providing computers with a vision system that could interpret visual cues and explain them was also tackled. In 1966, Seymour A. Papert and Marvin Minsky were assigning projects to undergraduate students at MIT and proposed the "Summer Vision Project" [125], were the summer workers were supposed to construct a significant part of a visual system. They postulated that it was a good task for the summer project because it can be segmented into sub-problems, which allow individuals to work independently. They were not wrong about the second part since this is still the way we currently tackle the task. However, the fact that large international research communities such as *robotics* and *computer vision* are still working on the problem represents a miscalculation of the timeline by several orders of magnitude.

In order to avoid the same type of miscalculation in this thesis, we do not tackle the entirety of the robotic vision problem that concerns us as a whole. Instead, we focus on how we can make several approaches that are already proven to work for a particular perception subtask, and find what we can modify to them in order to make them efficient enough for online operation in a mobile robot. While there is a whole field called *computer vision* which is responsible for solving the task proposed by Minsky and Papert, in the *mobile robotics* community we are interested in making the algorithms they develop efficient enough to run on our resource-constrained machines and fast enough to react to the changing environment. Naturally, this is not as dissociated as mentioned, and there is significant knowledge exchange between fields, with robotics researchers contributing seminal works to the other fields as well. In this thesis, however, we focus on the problem of efficiency in order to create algorithms that can run quickly and provide accurate results in low-power and low-compute hardware that can be mounted on a moving and sometimes even flying platform. Therefore, all of the approaches presented in this thesis follow one or several of the following premises in order to achieve this necessary efficiency: (i) use background knowledge about the task's domain in order to help the task, (ii) exploit properties of the sensor being used for each particular domain in order to process the data more efficiently, and (iii) pay close attention to the law of diminishing returns when adding compute or memory as a resource available to the algorithm, rather than chasing the last percent point in an evaluation metric. This is a concept from economy [158] that refers to a point at which the level of gains is less than the amount of resources invested, and which due to the aforementioned constraints needs to be constantly observed in our algorithmic design.

This thesis is divided into three parts. Part I focuses on the problem of scene understanding using camera images, and we present three algorithms that exploit background knowledge to do perception in agricultural fields and city environments efficiently, accurately, and on resource-constrained hardware. Part II focuses on the problem of scene understanding using LiDAR sensors. We present a large-scale dataset for the task, as well as two algorithms to solve it which also focus on efficiency. Finally, in Part III, we propose *one* possible way in which software for mobile robotics perception can be organized and developed and present our open-source frameworks based on this design paradigm. We implemented the presented frameworks with the main goal to allow for the deployment of our vision pipelines in dedicated, low-power hardware accelerators frequently used in the robotics community. This allows non-experts in computer vision to use our algorithms in a "plug and play" fashion on their robots, while also being able to effortlessly adapt them to their tasks.

1.1 Main Contributions

The main contributions of this thesis are efficient algorithms to perform different semantic scene understanding tasks and can run online on mobile robots. These algorithms serve as the first layer of processing after the sensor data enters the robot's computer and extract semantic information about the environment that is used downstream by other processes in the robotics software suite. Some examples of these downstream consumer tasks are: localization, mapping, obstacle avoidance, path planning, manipulation, surveillance, and monitoring, among others. Because of this, all of the perception algorithms presented in this thesis need to run fast on embedded hardware that is mounted on the robot. In Chapter 2, since the state of the art for most semantic scene understanding tasks is based in one way or another on deep neural networks, we also provide an introduction to them. Having a basic understanding of deep neural networks (DNN), and in particular convolutional neural networks (CNNs) is key in order to understand and evaluate our contributions. The fact that these deep neural networks are usually computationally expensive to run collides with our need for runtime efficiency, which motives our efforts to exploit knowledge about the underlying data distribution to aid the otherwise end-to-end pipeline proposed in the literature.

Part I of this thesis focuses on efficient semantic scene understanding using solely camera images. The first approach presented in Chapter 3 targets crop vs. weed classification in agricultural fields. This approach exploits an extra spectral cue from the near-infrared (NIR) spectrum, which contains useful information for agricultural tasks, for example, to extract region proposals for a posterior classification of the extracted regions, which makes the system more efficient and able to run online. Opposite to this, but still in the agricultural domain, Chapter 4 presents an approach to perform crop vs. weed semantic segmentation but without expensive to obtain NIR cues, and which relies solely on RGB images. Through the inclusion of extra input channels calculated from the RGB values, which can encode knowledge about the task at hand, we are able to bring back feature design into deep learning. This allows the architecture to be lightweight enough to run online on embedded hardware. Finally, Chapter 5 targets instance segmentation for city environments, which is relevant for autonomous driving. In this case, we also exploit background information about locally-connected areas in the image to operate in a lower-resolution grid, which makes the approach efficient,which makes this approach similar to both of the previous ones, albeit in a different domain.

Part II focuses on efficient semantic scene understanding but using solely point clouds obtained from LiDAR scans, which is of utmost relevance in the context of autonomous driving. In Chapter 6, we present SemanticKITTI, the first large-

scale dataset for semantic and panoptic segmentation of sequential LiDAR scans, with over 40 000 scans containing point-wise annotations for 28 classes. This dataset enabled both, the state of the art analysis as well as the developments presented in the subsequent chapters. Chapter 7 presents a thorough comparison of state-of-the-art approaches to semantic segmentation of point clouds. It also presents an efficient projective semantic segmentation architecture, which exploits an efficient proxy representation as an index to search for neighboring information and set a new state of the art for the task. Building on this work and exploiting the panoptic labels included in SemanticKITTI, in Chapter 8 we present the first two baselines to the panoptic segmentation of automotive LiDAR, as well as a novel, single-stage pipeline which achieves comparable results, albeit at a fraction of the runtime. As we mentioned several times already, this is key for robotics, and is particularly true for autonomous vehicles.

Finally, Part III presents an effective way to implement our semantic scene understanding frameworks, as well as an overview of how to use and extend their functionalities. Although Chapter 9 does not present a new research contribution per se, we believe it is mission-critical for researchers to write and share good code that they can scale, share, and also re-use, allowing us to make better science as a community effort. This is key for the independent validation of published results, as well as research findings. Thus, we present here our open-source developments that have also received a considerable uptake in the robotics community. Furthermore, in Chapter 10 we present four example use-cases of our open-source frameworks that allowed researchers in robotics to easily bring semantic information into their pipelines.

Overall, this thesis presents seven different contributions, ranging from the extraction of semantic information in agricultural fields and city environments using RGB images, to the release of a large scale dataset for LiDAR-only semantic and panoptic segmentation, along with two new state-of-the-art approaches for both tasks. Finally, we also presented three open-source frameworks for semantic scene understanding in robotics, and we explain their composition so that this document can serve as a starting point for the reader to start implementing on top of them or even to implement their own from scratch, according to their needs.

1.2 Publications

Parts of this thesis have been published in the following peer-reviewed conference and journal articles, or are currently under review:

- A. Milioto, P. Lottes, and C. Stachniss. Real-time blob-wise sugar beets vs weeds classification for monitoring fields using convolutional neural networks. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2017

- A. Milioto, P. Lottes, and C. Stachniss. Real-time Semantic Segmentation of Crop and Weed for Precision Agriculture Robots Leveraging Background Knowledge in CNNs. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018

- P. Regier, A. Milioto, P. Karkowski, C. Stachniss, and M. Bennewitz. Classifying Obstacles and Exploiting Knowledge about Classes for Efficient Humanoid Navigation. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (HUMANOIDS)*, 2018

- A. Milioto, L. Mandtler, and C. Stachniss. Fast Instance and Semantic Segmentation Exploiting Local Connectivity, Metric Learning, and One-Shot Detection for Robotics. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019

- A. Milioto and C. Stachniss. Bonnet: An Open-Source Training and Deployment Framework for Semantic Segmentation in Robotics using CNNs. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019

- L. Zabawa, A. Kicherer, L. Klingbeil, A. Milioto, R. Topfer, H. Kuhlmann, and R. Roscher. Detection of single grapevine berries in images using fully convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019

- A. Milioto, I. Vizzo, J. Behley, and C. Stachniss. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation. In *Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2019

- X. Chen, A. Milioto, E. Palazzolo, P. Giguère, J. Behley, and C. Stachniss. SuMa++: Efficient LiDAR-based Semantic SLAM. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2019

- J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding

of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019

- A. Milioto, J. Behley, C.S. McCool, and C. Stachniss. LiDAR Panoptic Segmentation for Autonomous Driving. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020. *Under Review.*

- J. Behley, A. Milioto, and C. Stachniss. A Benchmark for LiDAR-based Panoptic Segmentation based on KITTI. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020. *Under Review.*

- P. Regier, A. Milioto, C. Stachniss, and M. Bennewitz. Classifying Obstacles and Exploiting Class Information for Humanoid Navigation Through Cluttered Environments. *The Int. Journal of Humanoid Robotics (IJHR)*, 17(02):2050013, 2020

The following are publications I was involved in during my doctorate, but which are not part of this thesis:

- K. Franz, R. Roscher, A. Milioto, S. Wenzel, and J. Kusche. Ocean eddy identification and tracking using neural networks. In *Proc. of the IEEE Int. Geoscience and Remote Sensing Symposium (IGARSS)*, 2018

- P. Lottes, J. Behley, A. Milioto, and C. Stachniss. Fully convolutional networks with sequential information for robust crop and weed detection in precision farming. *IEEE Robotics and Automation Letters (RA-L)*, 3:3097–3104, 2018

- P. Lottes, J. Behley, N. Chebrolu, A. Milioto, and C. Stachniss. Joint Stem Detection and Crop-Weed Classification for Plant-specific Treatment in Precision Farming. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018

- X. Chen, T. Laebe, A. Milioto, T. Roehling, O. Vysotska, A. Haag, J. Behley, and C. Stachniss. OverlapNet: Loop Closing for LiDAR-based SLAM. In *Proc. of Robotics: Science and Systems (RSS)*, 2020

- P. Lottes, J. Behley, N. Chebrolu, A. Milioto, and C. Stachniss. Robust joint stem detection and crop-weed classification using image sequences for plant-specific treatment in precision farming. *Journal of Field Robotics (JFR)*, 37:20–34, 2020

- R. Sheikh, A. Milioto, P. Lottes, C. Stachniss, M. Bennewitz, and T. Schultz. Gradient and log-based active learning for semantic segmentation of crop

and weed for agricultural robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020

- A. Pretto, S. Aravecchia, W. Burgard, N. Chebrolu, C. Dornhege, T. Falck, F. Fleckenstein, A. Fontenla, M. Imperoli, R. Khanna, F. Liebisch, P. Lottes, A. Milioto, D. Nardi, S. Nardi, J. Pfeifer, M. Popović, C. Potena, C. Pradalier, E. Rothacker-Feder, I. Sa, A. Schaefer, R. Siegwart, C. Stachniss, A. Walter, W. Winterhalter, X. Wu, and J. Nieto. Building an Aerial-Ground Robotics System for Precision Farming. *IEEE Robotics and Automation Magazine (RAM)*, 2020

- F. Langer, A. Milioto, A. Haag, J. Behley, and C. Stachniss. Domain Transfer for Semantic Segmentation of LiDAR Data using Deep Neural Networks. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020. *Under Review.*

1.3 Collaborations

Some work presented throughout this thesis was performed in collaboration with colleagues at the Photogrammetry and Robotics laboratory at the University of Bonn, where I conducted all of my research work. The work presented in Chapter 3 and Chapter 4 of was done with the collaboration of Philipp Lottes, who provided the data from his recordings during the Flourish Project [90], using the Bosch DeepField Bonirob UGV. These works, albeit of my authorship, were supported by him through helpful discussions.

The work presented in Part II was mostly performed in collaboration with Jens Behley, who is a post-doctoral researcher at the laboratory. Besides being of invaluable help throughout my doctorate, Jens provided me with the labels for the LiDAR tasks. This meant both, labeling data by hand, as well as coordinating a team of student assistant labelers for the good part of a year, with the help of Martin Garbade.

Finally, all works presented in Chapter 10 as possible applications of my algorithms were lead by other doctoral students. My contribution in these works was mostly to collect the training data, train the models, provide the model architectures and weights to infer the semantic labels, and/or help set up the framework on the robot, as well as providing ideas on how to make the approaches communicate with the machine learning pipeline in an efficient way, which is my main interest. This is why this chapter is presented as an illustration of the usefulness of my three open-source frameworks, and not as a research contribution done specifically by me.

Chapter 2

Basic Techniques

We motivated in Chapter 1 why it is key for mobile robots to have semantic scene understanding capabilities. We also discussed different types of semantic scene understanding as we introduced the work developed in each chapter, and said that all of the state-of-the-art approaches to these tasks are in one way or another based on deep convolutional neural networks. This is also the case for all of the algorithms presented throughout this thesis, and therefore a basic understanding of them is key to understanding our contributions, which is why we focus this chapter on achieving that task.

It is very important to notice that before convolutional neural networks started to be used in practice for perception tasks in the last decade, a battery of other approaches existed to approach each specific task, and if this is the case, they will be properly referenced in each chapter's related work section. Those approaches, which took decades to develop, will not be ignored. However, the fact that neural networks can approximate any function to arbitrary accuracy [62] given that enough neurons are available, makes them attractive to tackle a wide variety of perceptual problems. This is, of course, added to the fact that over the last decade, the interconnected world has allowed us to amass an unprecedented amount of data, which can be used to train these algorithms to model these type of functions which we cannot, ourselves, describe mathematically. That being said, just because neural networks theoretically "can" represent any mathematical function, it doesn't mean that they can easily be trained to learn to represent said function. Therefore, in recent years, traditional computer vision approaches have made a come-back to assist this process, which is a recurrent topic through this thesis.

Note that the terms "neuron", as well as "neural network" also have analogous definition in biology. We will not go into detail about the analogy between biological neurons and neural networks and the artificial ones we use, and for

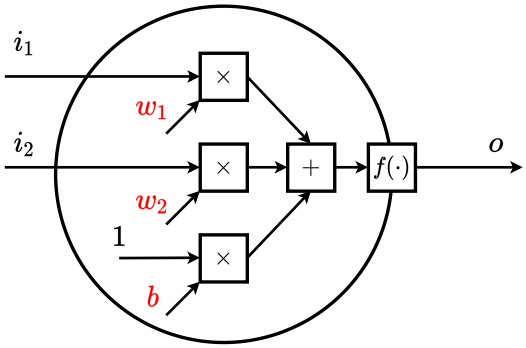

Figure 2.1: A simple neuron with two inputs, one output, and one non-linearity, calculating $o = f(i_1 w_1 + i_2 w_2 + b)$. $f(\cdot)$ is the non-linearity of the neuron, while w_1, w_2, and b, in red, are its parameters.

simplicity, from now on, whenever we refer to a neuron we refer to the artificial kind. In the following sections, we proceed to introduce what neural networks are, as well as a suite of techiques usually referred to as "deep learning", which allow us to train them to fullfil our desired tasks.

Before we can go into what a neural network is, we need to define its most basic unit: the neuron. Figure 2.1 shows a simple neuron which takes two inputs, multiplies them by its internal weights, adds a bias, and applies a non-linear function, before returning the output. This means that the output of the neuron is of the form:

$$o = f(i_1 w_1 + i_2 w_2 + b), \tag{2.1}$$

with i_1 and i_2 being the inputs to the neuron, w_1, w_2, and b its parameters, and $f(\cdot)$ its non-linearity. Such non-linearity can take a variety of different forms, with each one producing different properties in the output, and being adequate for different types of problems. Some examples of non-linearities, among many others, are:

- $\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$,
- $\text{ReLU}(x) = \max(0, x)$,
- $\text{LeakyReLU}(x) = \max(\alpha\, x, x)$,
- $\text{Tanh}(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, etc.

By tuning it weights and bias, and because of its non-linearity, which in all these cases is non-parametric, although parametric ones exist, we can represent a simple non-linear function. However, the representational capability of this simple neuron is very limited, and it is the composition of such neurons what makes neural networks powerful in their representation capabilities, as we will see in the following section.

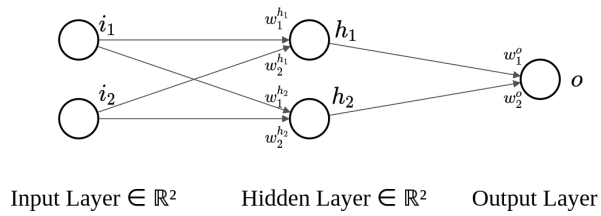

Figure 2.2: A simple feed-forward neural network with two inputs i_1 and i_2, two hidden neurons h_1 and h_2, and one output neuron o.

2.1 Fully-Connected Neural Networks

As we said in the previous section, the power of neural networks to represent complex, non-linear functions comes from the composition of these very simple units called neurons. Therefore, a neural network is defined as a collection of neurons interconnected using a certain topology. The simplest topology is the "feed-forward" neural network, such as is the one shown in Figure 2.2. Here, each of the neurons is of the type defined in Figure 2.1, and the topology is defined as a directed acyclic graph, such that there is no way to start at a neuron h_x and walk a path in the topology that brings me back to said h_x. Several other types of topologies exist, but are outside the scope of this basic introduction, and are not used in this thesis.

This forward connection pattern allows us to define the path from the inputs to each output as a composition of functions, which will be useful during the training. This means that the output can be calculated as:

$$o(h_1, h_2) = f(w_1^o\, h_1 + w_2^o\, h_2 + b^o), \text{with} \tag{2.2}$$
$$h_1(i_1, i_2) = f(w_1^{h_1}\, i_1 + w_2^{h_1}\, i_2 + b^{h_1}), \text{and} \tag{2.3}$$
$$h_2(i_1, i_2) = f(w_1^{h_2}\, i_1 + w_2^{h_2}\, i_2 + b^{h_2}). \tag{2.4}$$

To illustrate how such network weights can be adjusted to fit a desired function, we will use the simple feed-forward network defined in Figure 2.2, but the networks can be arbitrarily large and complicated, with thousands of inputs, hundreds of hidden layers, and thousands of outputs, which looks a lot more densely connected such as the network shown in Figure 2.3. This figure shows a network with 5 hidden layers, and 5 neurons in each. The number of hidden layers in a neural network is usually called its *depth*. Analogously, the number of neurons in one of these layers is called the layer *width*. These are concepts that show up in several points in this thesis, so it is worth to make a mental note of them.

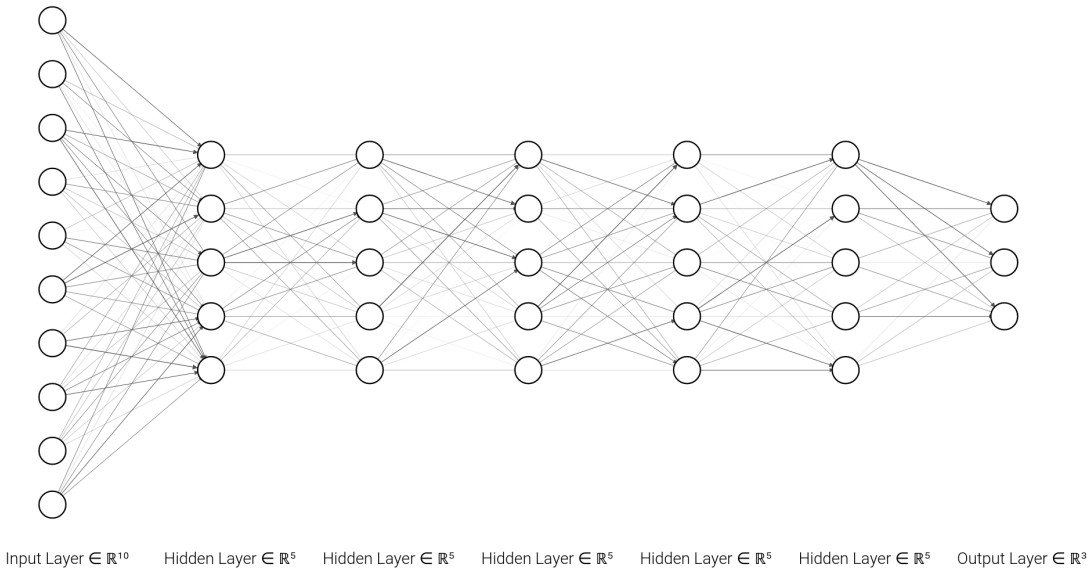

Figure 2.3: A more densely populated feed-forward neural network with ten inputs i_1 to i_{10}, 25 hidden neurons in 5 hidden layers, and 3 output neurons o_1 to o_3. In this representation, the intensity of the line represents the value of the weight, rather than giving it an explicit w value which would clutter the plot. Note that it is simple to change the number of inputs to any particular neuron (or its connectivity to the previous layer), by simply setting the corresponding weight to zero. We will discuss more about this later.

2.2 Training Neural Networks

Now that we know how a fully-connected, feed-forward neural network is set up, we can focus on actually making it represent the function that we are trying to model. This means adjusting its weights and biases, making it give the desired output for a set of input vectors for which we know the desired behavior. To achieve this, we define a cost, or loss function L, which depends on the task we are trying to solve, and we show the network what the output should be for every input vector, through the minimization of this cost. For the purpose of this illustration we assume that the output o from Equation (2.2) is regressing a continuous variable, and we use a mean-squared-error (MSE) loss to evaluate how far we are from the result at each step. Therefore, we obtain o_{pred} from each input vector $\mathbf{i} = (i_1, i_2)$ using Equation (2.2), and compare it with the desired output o_{gt} by using the loss:

$$L_{\text{MSE}}(\mathbf{w}, \mathbf{b}, \mathbf{i}) = (o_{gt} - o_{pred}(\mathbf{w}, \mathbf{b}, \mathbf{i}))^2, \tag{2.5}$$

where $\mathbf{w}$ and $\mathbf{b}$ are the vectors representing the model weights and biases. For our particular case, $\mathbf{w} = (w_1^o, w_2^o, w_1^{h_1}, w_2^{h_1}, w_1^{h_2}, w_2^{h_2})$, and $\mathbf{b} = (b^o, b^{h_1}, b^{h_2})$.

2.2.1 Gradient Descent

As Equation (2.5) shows, the three things we can change in order to make the loss smaller are (i) the inputs $\mathbf{i}$, (ii) the network's neuron weights $\mathbf{w}$, and/or (iii) its biases $\mathbf{b}$. Since the input is fixed by us, the only way to decrease the loss is by modifying $\mathbf{w}$ and $\mathbf{b}$.

The function described by the neural network is a composition of non-linear functions and results in a non-convex loss landscape. This means that the loss will likely have saddle-points and a large number of local-optima, making global optimization impossible. Therefore, when using neural networks the loss is minimized by using a variant of gradient descent, which given the non-convex nature of the problem, will converge to one of the multiple local-optima. Gradient descent is an optimization technique which walks the loss landscape locally and iteratively in the opposite direction of the gradients, which is the direction that minimizes the loss. Mathematically, this means that at each step of this iterative procedure, each of the network's weights $\mathbf{w}$ and biases $\mathbf{b}$ are updated by the rules:

$$w_{t+1} = w_t - \lambda \frac{\partial L}{\partial w}\Big|_{w_t}, \text{and} \tag{2.6}$$

$$b_{t+1} = b_t - \lambda \frac{\partial L}{\partial b}\Big|_{b_t}, \tag{2.7}$$

where $\frac{\partial L}{\partial w}\big|_{w_t}$ and $\frac{\partial L}{\partial b}\big|_{b_t}$ are the partial derivatives of the loss with respect to each w and b respectively evaluated at the current point in the loss landscape, and λ is the learning rate, which defines the speed at which gradient descent walks the loss surface, and is the most important hyper-parameter to select when training neural networks. The entire process starts with the selection of a random initialization of w and b, followed by multiple iterations of Equation (2.6) until convergence to one of the multiple function minima.

2.2.2 Backpropagation

As shown in Equation (2.6), at each iteration of the optimization through gradient descent, we need the gradients of the loss with respect to each w_i in $\mathbf{w}$ and b_i in $\mathbf{b}$. When dealing with neural networks, the process of obtaining such gradients is called "backpropagation" [157], which achieves this task *efficiently*, when compared with a naive calculation of the gradients with respect to each w_i and b_i individually. The backpropagation algorithm works efficiently in feed-forward neural networks because it computes the gradient of L with respect to each w_i and b_i one layer at the time by using the chain rule, walking backwards the same operations that it did forward. This avoids redundant calculations of intermediate terms that would be re-calculated if using a naive implementation of the

gradient calculation. It is this backward-walk style calculation which gives backpropagation its name, since it literally propagates the errors backwards. This reusing of intermediate calculations which provides an efficiency gain over the naive implementation is an example of dynamic programming.

To demonstrate how backpropagation works, we will calculate one of the gradients of the weights in the network defined in Figure 2.2. Let's calculate the gradient of Equation (2.5) with respect to the first weight in the first neuron of the hidden layer $w_1^{h_1}$. With $L_{\text{MSE}} = (o_{gt} - o_{pred})^2$, we can calculate the derivative with respect to $w_1^{h_1}$ by applying the chain rule:

$$\frac{\partial L_{\text{MSE}}}{\partial w_1^{h_1}} = \frac{\partial L_{\text{MSE}}}{\partial o_{pred}} \frac{\partial o_{pred}}{\partial h_1} \frac{\partial h_1}{\partial w_1^{h_1}}. \tag{2.8}$$

Expanding the first term:

$$L_{\text{MSE}} = (o_{gt} - o_{pred})^2, \text{therefore} \tag{2.9}$$

$$\frac{\partial L_{\text{MSE}}}{\partial o_{pred}} = \frac{\partial (o_{gt} - o_{pred})^2}{\partial o_{pred}} = -2\,(o_{gt} - o_{pred}). \tag{2.10}$$

Now expanding the second term:

$$o_{pred} = f(w_1^o\, h_1 + w_2^o\, h_2 + b^o), \text{therefore} \tag{2.11}$$

$$\frac{\partial o_{pred}}{\partial h_1} = \frac{\partial f(w_1^o\, h_1 + w_2^o\, h_2 + b^o)}{\partial h_1} = w_1^o\, f'(w_1^o\, h_1 + w_2^o\, h_2 + b^o), \tag{2.12}$$

where $f'(\cdot)$ is the derivative of the non-linearity applied to the argument, regardless of the non-linearity used. Finally, expanding the third term:

$$h_1 = f(w_1^{h_1}\, i_1 + w_2^{h_1}\, i_2 + b^{h_1}), \text{therefore} \tag{2.13}$$

$$\frac{\partial h_1}{\partial w_1^{h_1}} = \frac{\partial f(w_1^{h_1}\, i_1 + w_2^{h_1}\, i_2 + b^{h_1})}{\partial w_1^{h_1}} = i_1\, f'(w_1^{h_1}\, i_1 + w_2^{h_1}\, i_2 + b^{h_1}). \tag{2.14}$$

Piecing everything back together using Equation (2.10), Equation (2.12), and Equation (2.14) we get to the required gradient:

$$\frac{\partial L_{\text{MSE}}}{\partial w_1^{h_1}} = -2\,(o_{gt} - o_{pred})\, w_1^o\, f'(w_1^o\, h_1 + w_2^o\, h_2 + b^o)\, i_1\, f'(w_1^{h_1}\, i_1 + w_2^{h_1}\, i_2 + b^{h_1}), \tag{2.15}$$

which we can use to backpropagate the error and update $w_1^{h_1}$ using gradient descent. Note that because we walk the graph backwards, Equation (2.15) is not really the way this gradient is calculated. Instead, Equation (2.8) is calculated sequentially, pre-multiplying from the left, and forgetting the previous buffers as we walk back through the graph. This also means that some variables that can be reused, are reused, as stated at the beginning of this section, in a dynamic programming paradigm.

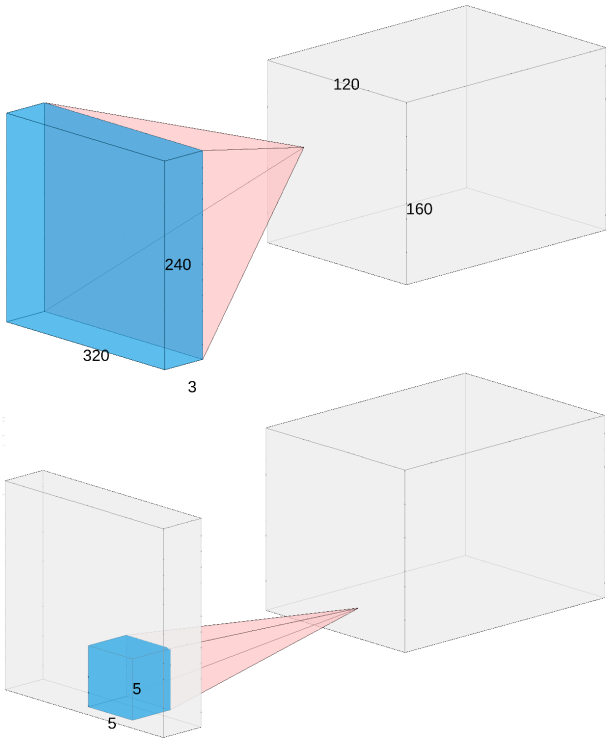

Figure 2.4: Difference between a neuron with a global receptive field (top), and a neuron with a local receptive field (bottom), in this case with a receptive field of 5×5.

2.3 Neural Networks for Image Processing

Up to now we have focused exclusively on inputs which were 1-dimensional vectors. E.g., in Figure 2.2, the input vector belonged to $\mathbb{R}^2$, and in Figure 2.3, it belonged to $\mathbb{R}^{10}$. However, the focus of this entire thesis is acting on either images, or image-like representations of LiDAR point clouds.

Although it is technically possible to reshape the input images from shape $H \times W \times 3$ into vectors in $\mathbb{R}^{H \times W \times 3}$, two problems arise from training fully-connected neural networks on this type of data. First, for even a small size image, e.g., 320×240 RGB pixels, which is a reasonable size for a lot of robotic applications, the initial layer of a network like the one in Figure 2.3, in which each neuron observes each input, would have over 1M parameters just in the first hidden layer. Second, each neuron in the network will be assigning equal importance to every portion of the image, ignoring the fact that closer pixels in the image are more closely correlated than further ones, and making relatinships between far away pixels that can confuse the training. Two solutions can be applied to solve this problem, both of which involve constraining the connectivity of each neuron with its previous layer, in two different ways, but equality depicted in Figure 2.4.

Locally-Connected Neural Networks. The easiest way to solve this problem is by simply dropping connections between a layer and its previous one. In Figure 2.3 we explained that we could drop connectivity by just zeroing the corresponding weights to the connection. While mathematically equal, in practice, this is done by avoiding the operation alltogether, since multiplying by zero would be a waste of computation. This partially solves the problem of having too many parameters in the first layer. It also addresses the problem of giving the same importance to points which are close to the ones that are far away. However, one caveat still remains: most of the high frequency features in an image can be detected at any position in the image, which means that we will likely learn duplicated kernels which detect the same feature but at different positions in the image. Furthermore, and an object for which we desire to observe the same feature, for example, a window in any part of the image, or the wheel of a car, will not be guaranteed to trigger with the exact strenght, even if the kernels which are duplicated through the effect previously mentioned are very similar. This desired property is called translation equi-variance, which means that a certain property in the image will trigger the same activation no matter where it is in the image. This means that not only the feature is detected with the same strength, but also its spatial relationship to other features is preserved, which allows deeper layers of the neural network to combine them to recognize more complex patterns composed of multiple image properties and their spatial relationships.

Convolutional Neural Networks. To solve the problems of locally connected layers not guaranteeing translation equivariance, LeCun *et al.* [87] proposed to force a set of units which act at different receptive fields of the image to share the same weights, forming 2-dimensional activation planes. This effectively turns the locally-connected layer into a convolution, which is the basis of all modern neural networks operating on images. Furthermore, this idea of sharing weights along some dimentions of the inputs, forming translation equivariant activation maps has been applied to other domains, and now sets the state of the art in speech recognition, text parsing, language translation, among many other tasks.

We believe the reader is now equiped with the tools necessary to understand both, how each one of our approaches work, and where our contributions lie within them.

Part I

Scene Understanding using Camera Images

Chapter 3

Crop vs. Weed Classification for Agricultural Robots

One of the most relevant use cases of perception in robotics in the last years has been in the field of agriculture robotics, to eliminate, or at least reduce, the amount of chemicals vested into the fields. Agrochemicals can have several side-effects on the biotic and abiotic environment and may even harm human health. In conventional weed control, the whole field is treated uniformly with a single herbicide dose, spraying the soil, crops, and weeds in the same way. An alternative, more sustainable way to tackle this task is through mechanical weeding. Another alternative is selective spraying only the areas most affected by weeds, or even on a per-plant basis. This, however, is an incredibly labor-intensive task, making the practice considerably more expensive to execute than its monodose counterpart. Furthermore, as progress makes countries increasingly industrialized, finding labor for these types of tasks is quickly becoming a challenging endeavor.

This is where robots have the potential to make a measurable impact since the automation of this procedure has the potential of reducing costs, as well as drastically reducing the amount of manual labor required to perform a per-plant treatment. To achieve this, robots working in fields must acquire precise knowledge about the spatial distribution of the weeds and plants in the environment. One way to collect this data is through unmanned aerial vehicles (UAVs) in combination with unmanned ground vehicles (UGVs) equipped with vision sensors. This, however, requires an automatic classification system that can analyze the image data online and label the crops as well as the weeds in the acquired images. One could even go a step further and use the autonomous ground vehicle to execute the mechanical or laser-based elimination of individual weed plants directly.

In this chapter, we address the problem of classifying image content into crops and weeds. By exploiting georeferenced locations of the UAV or UGV used to collect the images, the location for each plant can be determined using a bundle adjustment procedure. Through a detection on a per-plant basis of sugar beets (crops) and typical weeds, we can select relevant information for the farmers or for instructing the agricultural UGV operating on the field. Given the 3D information and pixel-wise labeling of the images obtained with our approach, we can directly provide the location of every weed to a ground robot or a selective spraying tool used on a traditional GNSS-controlled tractor.

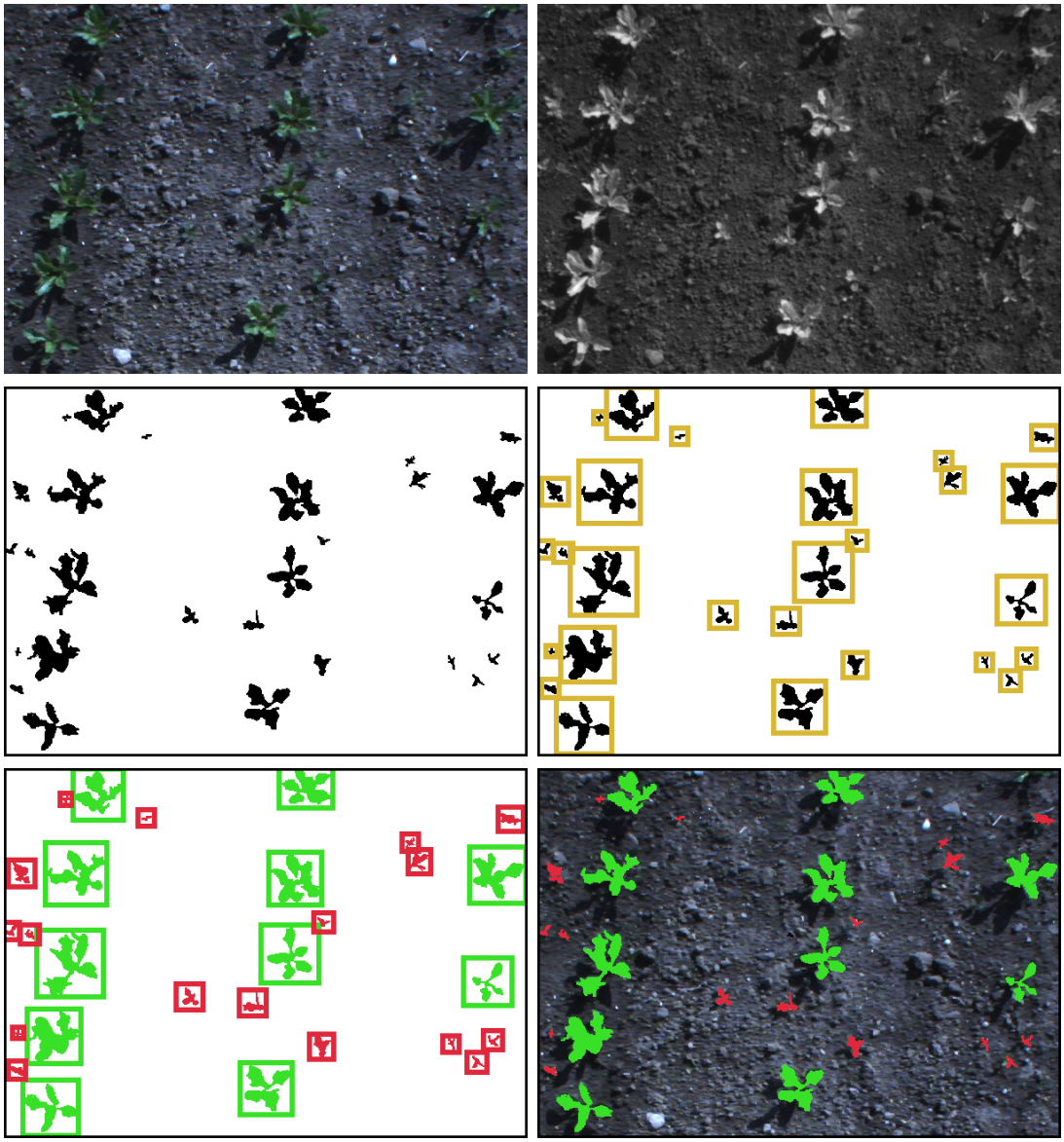

Figure 3.1: Illustration of our approach. Top: Original RGB and NIR images. Middle: NDVI-based mask and connected blob segmentation. Bottom: Classified blobs and pixel-wise overlay to original image.

3.1 Vision-based Crop vs. Weed Classification

In this section, we focus on the vision-based classification task of identifying sugar beet plants and weeds. To this end, we rely on RGB and near infra-red imagery of agricultural fields (see Figure 3.1 for an example). We use modern convolutional neural networks and deep learning pipelines to separate the plants based on their appearance. Our approach is an end-to-end approach, i.e., there is no need to define features for the classification task by hand. Our considered value crop is the sugar beet, which is an important crop in Northern Europe.

In most of the previous work on this topic [19, 38, 79, 100, 130, 181], instructions to extract features useful for the classification task need to be provided by an expert in the field. The main contribution of this work is a vision-based classification system to separate value crops from weeds using convolutional neural networks, where no hand-crafted features are used. The only preprocessing step we perform separates vegetation from the background such as soil, etc. We then only focus on the image regions containing vegetation. These sub-images are scaled to a standard size and fed into the neural network. As we show in this work, there is no need for the network to be especially deep to provide high-quality classification results for this task, and as a result, the amount of training data that is required to train the CNN is comparably small, in the order of a few hundred images. Furthermore, the network can be executed in real-time in edge-inference hardware small and efficient enough to fit in a UGV, and even a UAV, which is the main requirement to be useful for selective weeding.

We need to achieve a high recall on the weed classification to enable the robot to remove a high number of weeds on the field. We compute the coordinates relative to the robot's coordinate system to synchronize the removal with the extraction/spraying systems. This means that we are relaxed on the boundaries of the plants as long as we can direct the removal tools to the right position within our desired accuracy without harming the valuable crops. This allows for certain hypotheses that make a computationally cheap and fast blob approach suitable while still achieving state-of-the-art performance, such as: doing most of the training of the classifier and weed removal in early stages of the crop growth, and relaxing on the precision of the boundaries of the vegetation at the pixel level. As we can see in the last image of Figure 3.1, where the output classification is overlayed with the input image, a combination of the NDVI thresholded segmentation with a blob segmentation and classification gives a solid almost pixel-wise representation of the output mask, due to the masking done by the preprocessing, making this algorithm not only faster but also very similar in output quality as the pixel-wise approach, which is both computationally more complex and more time expensive to label.

Figure 3.2: From left to right: RGB and NIR images, NDVI transform, and mask obtained by a conservative threshold in the NDVI image.

We divide our classification pipeline into three main steps. First, we perform a vegetation detection to eliminate irrelevant soil information from the images using multispectral images containing RGB and near infra-red information. Second, we describe the detected vegetation with a binary mask and perform a blob segmentation to extract patches containing singular crops or weeds. Finally, we classify each patch with a convolutional neural network that we train in an early growth stage to avoid overlapping crops and weeds in the data, and that we can, later on, re-train in a supervised or semi-supervised manner to adapt to other types of crops, or other stages of the crop growth. The blob approach makes the network run fast for both the forward pass and the retraining of the classifier, which makes it able to run on a flying platform using readily available single-board computers such as the NVIDIA Jetson platform.

In sum, we make the following four key claims: Our approach (i) does not require any hand-crafted features and thus can be implemented quickly, (ii) can provide accurate weed classification in real sugar beet fields with an accuracy of more than 95% in a field with plants in similar growth stage, (iii) can be retrained efficiently to reuse the learned features for other data types or later growth stages, and (iv) is lightweight and fast, which makes it suitable for online operation in a moving and even a flying robot.

In the following subsections, we discuss each step of the pipeline in detail.

3.1.1 Vegetation Detection

To feed the CNN with interest regions, we perform a pre-processing step based on the extra NIR cue. The goal of this pre-processing step is to separate the relevant vegetation content of every image $\mathcal{I}$ taken by the robot from the soil beneath it, for which we need to apply a mask of the form:

$$\mathcal{I}_{\mathcal{V}}(i,j) = \begin{cases} 1, & \text{if } \mathcal{I}(i,j) \in \textit{vegetation} \\ 0, & \text{otherwise} \end{cases}, \tag{3.1}$$

with the pixel location (i, j).

As we can see in Figure 3.2, separating vegetation from soil in a plain RGB image is difficult task even for a human observer. Luckily, an approach exploiting the high reflectance of chlorophyll in the near infra-red spectrum has been thoroughly studied by Rouse *et al.* [155] prompting us to use the Normalized Difference Vegetation Index (NDVI), which is a normalized ratio of the NIR and red bands. If we apply this index pixel-wise we obtain a distribution that is more suited for a thresholded segmentation than the original image, which we can later apply as our mask, as we can also see in Figure 3.2. Namely, using the NDVI to create the mask we have:

$$\mathcal{I}_{\mathrm{NDVI}}(i,j) = \frac{\mathcal{I}_{\mathrm{NIR}}(i,j) - \mathcal{I}_{\mathrm{R}}(i,j)}{\mathcal{I}_{\mathrm{NIR}}(i,j) + \mathcal{I}_{\mathrm{R}}(i,j)} \tag{3.2}$$

Such transformation is used successfully to solve this very problem by Lottes *et al.* [99], and improved by Potena *et al.* [136] by adding a low cost convolutional neural network to a mapping obtained by permissive thresholding in the post NDVI segmentation operation. In our experiments, we use the thresholded segmentation of the NDVI-transformed image, similar to Lottes *et al.* [99], to separate the plants from the soil. An example is shown in Figure 3.2.

A threshold-based classification based on the NDVI can also lead to individual pixel outliers, or small groups of pixels. These are mostly caused by lens errors as chromatic aberration, which leads to a slightly different mapping of the red and the near infra-red light on the camera chip. We eliminate most of these effects through basic image processing techniques such as morphological opening and closing operations to fill gaps and to remove noise at contours, which is done with a 5×5 disk kernel, a 1× opening followed by a 1× closing; and removal of tiny blobs which share less than a minimum amount of vegetation pixels, which depends on the Ground Sampling Distance and therefore it varies with the UAV distance to the ground.

3.1.2 Blob-wise Segmentation

Given the vegetation mask, we search for vegetation blobs. Each vegetation blob is given through a connected component of the vegetation pixels in the image (see Figure 3.3). We create an individual image patch of a certain size (64×64 pixels in our current setup) for each detected blob in the vegetation mask. Here, we first compute the bounding box of the contour of the blob and apply an affine transformation to it to translate the blob into the center of the image patch and scale it to the desired size.

These image patches serve as input for our classification approach. Thus, the CNN can learn a representation based on such patches, which mostly describe whole plants or leaves. Through centering the blobs within the image patches we

achieve translation invariance and through scaling the detected blobs we reduce the influence of the plant size, which can vary significantly even for plants growing in the same field.

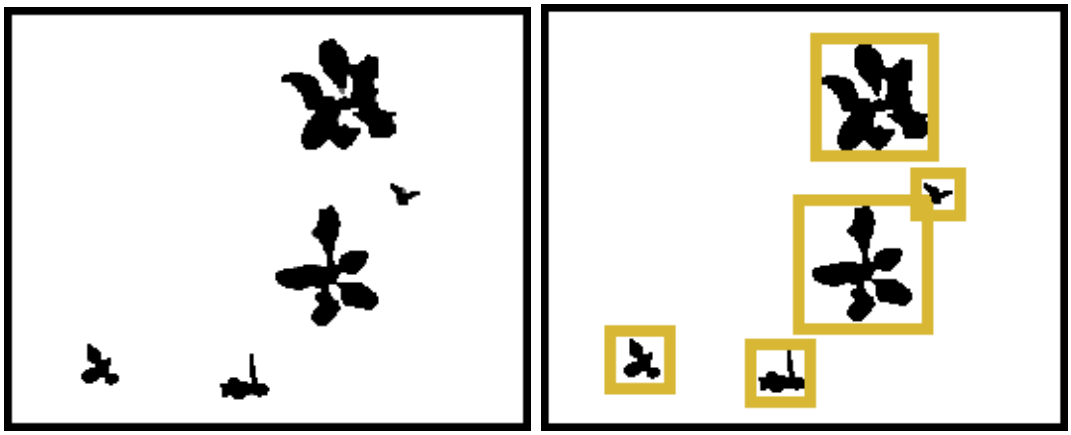

Figure 3.3: Blob selection from connected regions.

3.1.3 Classification using Convolutional Neural Networks

The advantage of using Convolutional Neural Networks for image classification, compared to traditional machine learning-based approaches, is that they can learn features describing complex non-linear relations between dependent and independent variables by themselves, when given enough data. This is not only useful for achieving a high classification accuracy in a specific dataset, but also to later transfer that knowledge to other domains. Therefore, we can train the network with a high number of plants from different species under different weather and lighting conditions and then use the feature extractors to cheaply train other classifiers for unseen species.

The main ideas behind weight sharing and CNNs were introduced by LeCun *et al.* in [86] and [87], and more recently brought back to public interest by Krizhevsky *et al.* [78] where they exploited an efficient GPU implementation of the convolution to break the ImageNet challenge benchmark[34] at the time by an astounding 10% difference over their closest competitor, making them widely popular. They make the assumption that in image data, neighboring pixels are more correlated than pixels far away, therefore introducing the concept of local connectivity, allowing for less neuron wiring, which yields not only in better results but also in less training and execution time. Today, deep convolutional neural networks are undoubtedly the most widely used approach for image classification, with its more novel flavors achieving top-5 classification errors in the test set of ImageNet as low as 3.57% [60], and 3.08% [169], outperforming the average human classification error which is around 5% on this data.

The original architecture used by Krizhevsky [78] is comparably shallow, i.e., it has fewer layers, when measured against today's state-of-the-art image classification networks, but we use a similar one to solve our classification problem, because it proves to have enough expressiveness to describe our decision boundaries and has the advantage that it is cheap to train and run in compact and low-power consumption hardware such as the one in a UGV, or UAV.

3.1.4 Selection of the Network Architecture

The network has to be complex enough to be able to describe the decision boundary between classes, but not too expressive as to overfit to our training data. This characteristic is set by the depth of the network, i.e., the number of convolutional layers, as well as its width, which represents the number of kernels in each convolutional layer or the number of neurons in the fully-connected ones. To choose our network size, we iterate starting with networks of a small number of filters per layer and only one convolutional layer and one fully connected, and grow the network until we are almost able to overfit to a small sample of our dataset. After this, we further increase the expressiveness by adding more kernels to each convolutional layer and more neurons to the fully connected ones, and we train it again using our full training dataset. To avoid the bigger network to overfit to our training data we use a method called "dropout" [164]. This prevents neurons from learning complex feature co-adaptations by randomly dropping some of them during training, making the network generalize better, which we corroborate by getting a good performance in our cross-validation data. It is important to notice that the versions of dropout used in convolutional and fully connected layers are different. For convolutional layers, whole channels represent activation maps of each feature, and therefore a special spatial dropout is employed, which drops entire feature maps, rather than individual neurons. We also have to be careful to set an upper boundary in the network's complexity, due to our constraint of real-time capability, meaning that the network needs to be both lightweight and of a complexity that is low enough to run fast in an embedded platform that we can fit in a flying vehicle. We are careful when choosing the depth and size of each layer to enforce this restriction.

Finally, we want our classifier to represent a vector containing a probability distribution of the input patch being a plant and a weed. For this, we use output neurons with a softmax activation function, which convert the output of the network to a pseudo-probability distribution that works well for this purpose.

From this, we design a network like the one in Figure 3.4. This architecture is composed by 3 convolutional layers (2 of kernel size 5×5 and the last one of 3×3 kernels), all of them using the Rectified Linear Unit for the non-linearity, which proves to speed up training considerably, and followed by 2×2 max-pooling

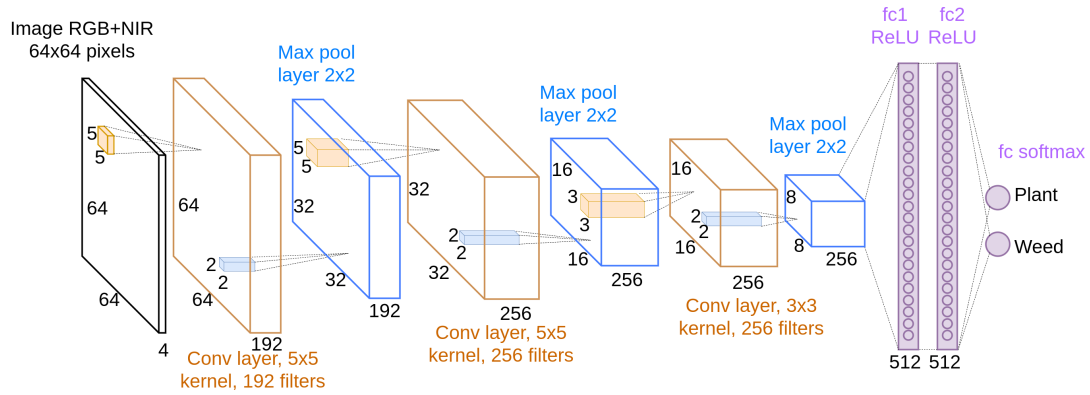

Figure 3.4: Detailed architecture used for the blob-wise classification.

for dimensionality reduction. These 3 convolutional and max-pooling layers are followed by 2 fully connected layers with 512 neurons each, which results in a 512-dimensional vector to be fed into the softmax layer that outputs the pseudo probability distribution we want as an output.

It is important to note that even though the network is shallow compared to the ones used by [60, 169], it is more complex and than those normally used for plant discrimination problems [136]. However, due to the blob approach, the network does not need to examine every pixel of the original input image, and it only runs one time per segmented patch, speeding up the process considerably.

3.1.5 Retraining for Different Crop Growth Stages

As the plants grow, they not only change their size but also their shapes and texture, which means that we may not get the best results if the growth stage is very different. In this case, the features extracted from the convolutional and fully connected layers may still be useful, so we can try to retrain the last layer of the network, which is the classifier, by feeding the network with a sample of labeled data in the new growth stage. We will show that even though the classifier is still usable without retraining, this technique can yield significant improvements, and can be done in a fast way due to the need to only train a linear classifier and in a small subset of the new data. This allows the farmer to provide a few training samples, and for the re-training of the classification layer to happen in the embedded hardware mounted on the platform.

3.2 Relationship to Object Detectors

The framework proposed in this work follows the same core ideas as the prohibitively expensive original RCNN method [49], performing a region-proposal

extraction, followed by a classification CNN. However, since the extra NIR information present in the input data allows us to quickly and accurately generate the blob proposals, this type of approach becomes feasible for online operation. This provides a good example of how domain-specific data can be exploited to improve on deep learning approaches, even without designing features to be used for the classification, since this vegetation segmentation can be exploited for an incredibly wide variety of plant species.

3.3 Experimental Evaluation

The experiments are designed to show the accuracy and efficiency of our method and to support the claims that our classifier is able to (i) learn feature representations by itself and thus be implemented quickly, (ii) learn an accurate decision boundary to accurately discriminate plants and weeds with an accuracy of over 95%, being able to eliminate a majority of weeds with a minimal number of misclassified crops when tested in a similar growth stage; (iii) quickly improve after retraining by using the previously learned features and adapting the last layer to a small subset of labeled new data; and (iv) run in real-time.

We implement the whole approach presented in this work relying on the Google TensorFlow [1] library. This allowed an outsider to the field of plant classification to perform the full implementation of the approach and its evaluation in slightly over two months, not including the acquisition time of the training and evaluation datasets, which supports our claim (i) that due to the self-learning of feature descriptors and the simplicity of the overall pipeline, the approach can be implemented quickly.

3.3.1 Training of the Network

To train the network we use a dataset which we will call "Dataset A", from Table 3.1. We split the dataset in 3, leaving 70% of it available to us for training, 15% for validation and 15% for test after the whole pipeline was designed and trained, and therefore show how the network generalizes to unseen data. This dataset was captured with the plants in an early growth stage and it is quite balanced in its plants/weeds relation. We compare it with Dataset B from Table 3.1, which was captured 2 weeks later, in a more advanced growth stage, and we analyze the performance of the network. Both datasets were captured using a JAI camera in nadir view capturing RGB+NIR images. For a typical robot-based monitoring scenario, not every image from the camera needs to be classified, and one image per second is sufficient. Thus, we take images at 1 Hz, setting the upper bound for testing our real-time classification capabilities to 1 FPS.

Parameter	Dataset A	Dataset B
no. images	867	1 102
no. patches	7 751	28 389
no. crops	2 055	2 226
no. weeds	5 696	26 163
P_c/P_w	0.36 / 0.64	0.08 / 0.92
avg. patch/image	9	25

Table 3.1: Information about the datasets.

Model	Cores	Architecture	TFLOPs	RAM
Titan Xp	3 840	Pascal	12	12GB
Jetson TX2 SoC	256	Pascal	1.5	8GB
GTX 940MX	384	Maxwell	0.84	2GB
Intel i7	-	-	0.04	-

Table 3.2: Hardware used for inference.

A commonly used technique for augmenting datasets is to apply affine transformations to the training portion of the dataset, for the network to learn some type of invariance. In our case, all the plants are previously scaled to patches of 64x64 pixels, aiming for scale and translation invariance, and we further apply 64 even rotations to try to learn rotationally invariant features. After this, we perform a per image normalization, which allows us to increase the robustness to illumination changes.

Then we run the training using a cross-entropy cost function, which we minimize by using stochastic gradient descent in batch sizes of 100. During training, we reduce the learning rate by a third every 2 epochs to converge better the cost function's minima, and we run until convergence is achieved.

The training takes 3 days on a low-cost NVIDIA GeForce GTX 940MX, which is not a top-notch graphics card and it is actually less powerful than a currently available embedded platform like the Jetson TX2 which runs on a drone. This is critical if re-training is necessary, as we show later. If we use state-of-the-art GPU's for training, the process is cut short by an order of magnitude, not only because of the difference in the floating operations per second but also because the whole pipeline can be stored in GPU memory avoiding a bottleneck in the copying operations from system memory to GPU memory before each calculation. In Table 3.2 we show these differences. This further supports our claim (i) that the system can be implemented quickly, given the short training time even in a low-cost GPU.

3.3.2 Performance of the Classifier

This experiment is designed to support our claim (ii) which states that we learn an accurate decision boundary to accurately discriminate plants and weeds with an accuracy of over 95%, being able to eliminate a majority of weeds with a minimal number of misclassified crops.

We now analyze the accuracy of the learned classifier and we evaluate its invariance to the crop growth. We tested the accuracy of our classifier in the test set which only contains unseen data of dataset A, and its accuracy in the test set of dataset B, when trained in A. We can see the results in Table 3.3:

Dataset	**Test set A**	**Test set B**
Accuracy	97.3%	89.2%

Table 3.3: Accuracy of the classifier trained in train set of A, and evaluated in Test set of A and B.

As we can see, we have a very high overall accuracy in the unseen data from dataset A, because it is from the same crop growth stage as the data the the network has seen while it was being trained, and the performance of the classifier drops when used 2 weeks later. However, the overall accuracy does not say much about the quality of the classifier, especially in cases like dataset B, that are highly unbalanced. Since the main goal of our classifier is to detect weeds to proceed to their elimination, in Table 3.4 we show the obtained precision and recall for the detection of weeds in both datasets.

Dataset	**Dataset A**	**Dataset B**
True Positives	829	16 353
True Negatives	290	1406
False Positives	27	94
False Negatives	16	2 020
Weed precision	96.84%	99.42%
Weed recall	98.10%	89.00%

Table 3.4: Quality comparison of the classifier for the weed class trained in A.

It is important to note that due to the nature of our problem it is mission-critical to have high precision in the weed classifier, because a large number of false positives means that plants get classified as weeds and thus get eliminated, especially in the case of mechanical treatment. This is not the case for selective spraying, but spraying plants with herbicides defeats the purpose of the system

anyway. It is also desirable to have a high recall to eliminate the most weeds we can, but not eliminating crops is of critical importance.

Taking this into consideration, we can see from Table 3.4 that even if the recall of the weed detector decreases two weeks later, the precision is very high, allowing us to not eliminate valuable crops erroneously, fulfilling our demands for the system, and supporting our claim (ii).

3.3.3 Retraining the Network

This experiment is designed to validate claim number (iii), i.e., that our system can be easily and cheaply retrained to adapt to new growth stages.

As stated in the previous section, even though the precision of the weed detector was still very high two weeks later in the growth stage, the recall fell considerably, meaning that a good number of weeds were not being removed from the field. To fix this problem, we can cheaply retrain the last layer of the classifier, re-utilizing the feature vectors extracted by the convolutional filters learned from dataset A. We show the results it yields in Table 3.5, where we can see that both precision and recall have improved, making the system not only eliminate fewer crops erroneously but also eliminate more weeds, which is the main purpose of it. In Table 3.6 we can also assess the general accuracy of the retrained classifier.

Dataset	**Trained in A**	**Retrained in B**
True Positives	16 353	17 927
True Negatives	1 406	1 190
False Positives	94	60
False Negatives	2 020	696
Weed precision	99.42%	99.66%
Weed recall	89.00%	96.26%

Table 3.5: Quality comparison of the classifier for the weed class in test set of B, before and after retraining.

Dataset	**Trained in A**	**Retrained in B**
Accuracy	89.2%	96.1%

Table 3.6: Accuracy in dataset test set of B before and after retraining on small subset (15%) of training set of B.

What is most important to point out is that, while the full classifier takes 3 days to train in the cheap GPU, the retraining achieves remarkable improvements in only 20 minutes. This is done by feeding the training with a balanced subsample (15%) of dataset B containing the same number of weeds and plants.

This supports our claim that our system can be easily and cheaply adapted to new growth stages where plants have significantly changed their appearance.

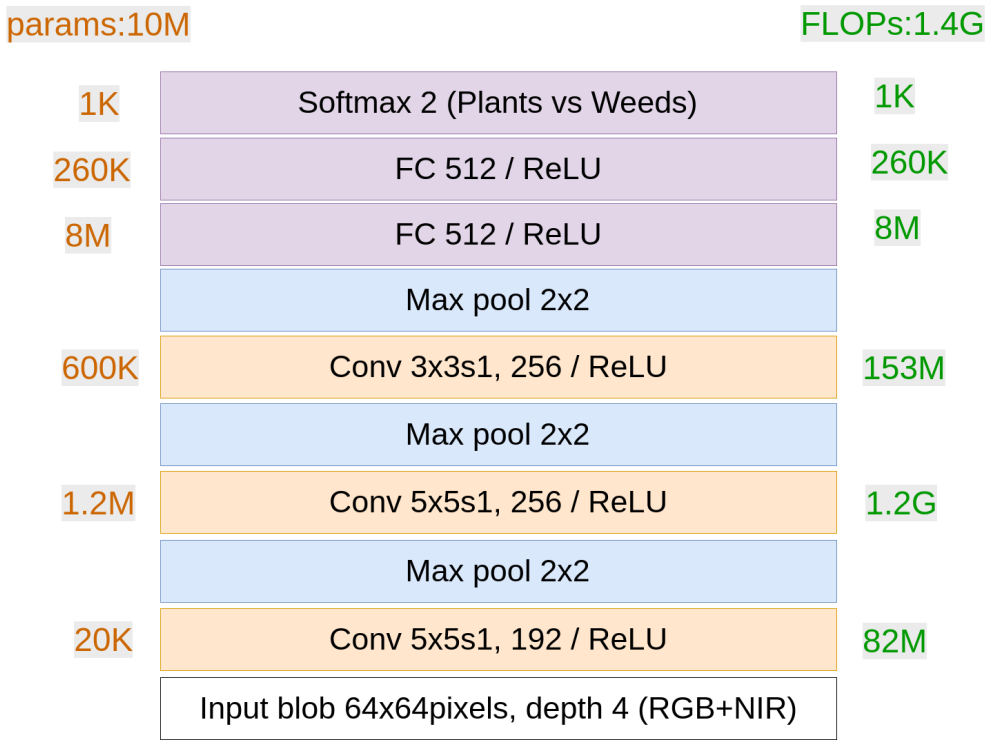

Figure 3.5: Architecture used for the blob-wise classification, with the size of each layer specified and the amount of floating operations needed per pass.

3.3.4 Runtime

This experiment is presented to validate claim (iv), i.e., that our system can run in real-time on a flying platform. In Figure 3.5, we show another representation of the network, in which we can also see how many variables each layer contains, and how many floating operations it takes to pass through it. The final architecture needs 1.4 G floating-point operations to do a full forward pass, which means that we can roughly calculate how long it will take to do a full pass depending on the FLOPs of the hardware. This results in Table 3.7. It is important to note that we left out the most powerful GPU because it is non-realistic to think that we would put a desktop workstation in a small flying drone with the purpose of inference, and it is also highly unlikely in a ground robot.

Another aspect we can see from Figure 3.5 is that the architecture is using a total number of 10 M 32-bit floating point parameters. This means that we can have a full network size of 40 MB, which is something that our embedded hardware can store easily.

In Table 3.8, we analyze how long it would take in average to do a forward pass

Hardware	Theoretical time	Actual time
Jetson TX2 SoC	1 ms	±1 ms
Geforce GTX 940MX	2 ms	7 ± 1 ms
i7 Processor	40 ms	50 ± 10 ms

Table 3.7: Average and standard deviation running time per blob, depending on the hardware.

in all patches from a raw image, depending on the dataset. This highly depends on the growth stage, as we can see from the fact that dataset B has more patches per image, meaning that there are more plants and weeds present in the field, which makes sense. In this table, we see that even with 25 images per image, which is our worst case, we are still very relaxed in our real-time constraint, by a margin of 5 in the case of our outdated GPU and a margin of 10 in the state of the art SoC, fulfilling our efficiency claim. This means that we can either analyze images at a higher frame rate, or fly higher, covering more crops, if the resolution of the camera is good enough to capture them from a distance. We also show that for our purposes, it is worth it to lose constant time complexity of the algorithm towards linear time because due to the upper bound of the number of patches contained in each raw image, we are actually gaining in performance.

Dataset	Dataset A	Dataset B
avg patches per image	9	25
GEForce940Mx(7 ms/patch avg)	63 ms	175 ms
Jetson TX2(4 ms/patch avg)	36 ms	100 ms

Table 3.8: Average runtime per image.

As a final comment about runtime, we should consider that for each raw input image containing n vegetation blobs we segment we get a patch array of size n. Since our neural network platform allows us to run forward passes in batch, we can gain a considerable increase in performance if we send the whole batch of blobs to the GPU RAM at the same time, rather than queuing all operations sequentially.

The presence of low cost, lightweight, power-efficient hardware that can run this classifier in remarkably low time with great performances, make this approach a great alternative for online use in moving robotic platforms working in dynamic environments. This supports our claim number (iv) that our system can run in real-time in a UAV or a UGV.

3.4 Related Work

The ability to estimate semantic information from sensor data is an important capability for autonomous and semi-autonomous systems operating in the fields [100]. Robotic approaches for monitoring key indicators of crop health have become a popular research topic. Geipel *et al.* [47], Khanna *et al.* [75], and Pfeifer *et al.* [131] estimate the crop height as well as the canopy cover using UAV imagery. These works rely on bundle adjustment procedures to compute terrain models and subsequently perform a vegetation segmentation to estimate the crop height based on the obtained 3D information.

Several approaches have been developed for vision-based vegetation detection by using RGB as well as multispectral imagery of agricultural fields [54, 57, 176]. Hamuda *et al.* [57] present a survey about plant segmentation in field images by analyzing several threshold- as well as learning-based methods. Torres Sanchez *et al.* [176] investigate an automatic and adaptive thresholding method for vegetation detection based on the Normalized Difference Vegetation Index and the Excess Green Index (ExG). They report a vegetation detection rate of around $90 - 100\,\%$. In contrast, Guo *et al.* [54] apply a learning-based approach using a decision tree classifier for vegetation detection. They exploit spectral features using different color spaces based on RGB images to perform a prediction on a per-pixel basis.

In the context of leaf image classification and segmentation, researchers have investigated using hand-crafted features and structural operators [181, 79, 19, 38]. Wang *et al.* [181] segment leaf images by using morphological operators and shape features and apply a moving center hypersphere classifier to infer the plant species. Kumar *et al.* [79] start from segmented images of leaves. They extract curvature features and compare them with a given database to find the best match with a labeled type. To cover a variety of leaf shapes, Cerutti *et al.* [19] apply a deformable leaf model and use morphological descriptors to classify their species. Elhariri *et al.* [38] compare a random forest classifier and a linear discriminant analysis based approach in their work for classifying 15 plant species by analyzing leaf images. They extract statistical features from the HSV color space of leaf images as well as gray level co-occurrence matrices to exploit additionally shape and vein (leaf structure) information. Hall *et al.* [56] conducted a study on features for leaf classification. They compared the classification performance based on classical hand-crafted and ConvNet features by using a random forest classifier. They simulated varying conditions on the public Flavia leaf dataset and conclude that CNN features support the performance and robustness of the classification results, which further motivates our approach.

We avoid the time-consuming process of designing features and exploit a CNN

approach to completely learn a representation of the data, which is suitable for discriminating sugar beets and weeds in image patches.

In the field of machine learning, several approaches have been applied to classify crops and weeds in the imagery of a plantation [44, 53, 130, 129, 58, 100, 102, 84]. Perez Ortiz *et al.* [129] propose an image patch-based weed detection system. They use pixel intensities of multispectral images and geometric information about crop rows to build features for the classification. They achieve overall accuracies of $75 - 87\%$ for the classification by analyzing the performance of different machine learning algorithms. Perez Ortiz *et al.* [130] use a support vector machine classifier for crop/weed detection in RGB images of sunflower and maize fields. They exploit statistics of pixel intensities, textures, shape, and geometrical information as features. Guerrero *et al.* [53] propose a weed detection approach for image data containing maize fields. Their method allows identifying the weeds after its visual appearance changed in image space due to rainfall, dry spell, or herbicide treatment. Garcia *et al.* [44] propose an approach for separating sugar beets and thistle based on narrowband multispectral image data. They apply a partial least squares discriminant analysis for the classification and obtain a recall of 84% for beet and 93% for thistle by using only four of the narrow bands at 521, 570, 610, and 658 nm. Haug *et al.* [58] present a method to detect carrot plants in weeds by using RGB and NIR-images without needing a pre-segmentation of the scenes into vegetation blobs. They perform a prediction on pixel level on a sparse grid in image space and report an average accuracy of around 94% on an evaluation set of 70 images where both, intra- and inter-row overlap is present. Lottes *et al.* [100, 102] design a crop and weed classification system for ground and aerial robots. The work exploits NDVI and ExG indices to first segment the vegetation in the image data and then applied a random forest classifier to the vegetative parts to further distinguish them into crops and weeds. Latte *et al.* [84] use color space features and co-occurrence matrices to classify images captured on crop fields containing several plant types. They apply an artificial neural network classifier and obtain an accuracy of around 84% on average. Their results show that using statistical moments of the HSV distribution improves the overall classification performance.

Also, deep learning methods have recently been employed to obtain plant statistics and distinguish value crops and weeds in the fields [116, 24, 105, 136]. Mortensen *et al.* [116] apply a deep CNN for classifying different types of crops to estimate individual biomass amounts. They use RGB images of field plots captured at 3 m above the soil, and report an overall accuracy of 80% evaluated on a per-pixel basis. Chen *et al.* [24] propose a visual system with the purpose of obtaining a robust count of fruits in the field under dramatic lighting changes, and with heavy occlusions from neighboring foliage. They first use a CNN for

blob region proposal, then they use a counting algorithm based on a second CNN to estimate the fruit count in each region, and they finally apply a linear regression model to get the final count. McCool *et al.* [105] address the crop and weed segmentation problem by using an end-to-end ensemble of CNNs. They obtain these lightweight CNNs through the compression of a pre-trained, more descriptive and complex model, and report an accuracy of 93.9%, processing images at little over 1 FPS. Potena *et al.* [136] present a perception system based on RGB+NIR imagery for crop and weed classification using two different CNN architectures. A shallow network performs the vegetation detection and then a deeper network further distinguishes the detected vegetation into crops and weeds. They perform a pixel-wise classification followed by a voting scheme to obtain predictions for detected blobs in the vegetation mask. They obtain a performance of around 97% for the vegetation detection, which is comparable to a threshold-based approach based on the NDVI, and 98% for the crop and weed classification in case the visual appearance has not changed between the training and testing phases.

Our approach uses RGB-NIR imagery and CNNs and can reach an accuracy of 97% with low running times even in embedded hardware that can be fitted in a moving robotic platform. We achieve 97% precision and 98% recall for the classification of the weeds when the classifier is used in a similar growth stage as the one it was trained in. We are also able to obtain 99% recall and 89% precision for weeds in later growth stages without retraining. Finally, we show that we can jump from 89% to 97% in recall for weeds if we retrain the classifier with new data from the current growth stage in a fast way. All of this allows our system to do statistics and selective weed removal in a very quick and accurate manner.

3.5 Conclusions

We have mentioned that UAVs and UGVs for precision farming must be able to distinguish the crops on the field from the weeds. In this chapter, we addressed the problem of detecting sugar beet plants and weeds using an RGB+NIR camera over real fields. We developed a classification system based on convolutional neural networks. Our approach is purely vision-based exploiting 4-channels and does not require geometric priors such as that the sugar beets must be planted in crop rows. In addition to that, the CNNs are an end-to-end approach such that no manual features must be defined. The input to the CNNs are blobs of vegetation and provide a classification output for such blobs. This renders the approach fast such that the system can be executed online. quickly in the order of 2 months. We implemented our approach and thoroughly evaluated it using image data recorded on a real farm in different sugar beet fields and illustrate that our approach allows for accurately identifying the weeds on the field.

Chapter 4

Crop vs. Weed Semantic Segmentation using RGB-Only Data

In Chapter 3 we formulated that herbicides and other agrochemicals which are frequently used in crop production can have several side-effects on our environment, and thus, one big challenge for sustainable approaches to agriculture is to reduce the amount of agrochemicals that needs to be brought to the fields. In the approach presented in that chapter, as well as in a large number of approaches in the literature, additional spectral cues such as near infra-red information are necessary. This is especially true for those approaches that should yield a high classification performance.

Often, these techniques also rely on a pre-segmentation of the vegetation (see Section 3.1.1) and/or on a large set of hand-crafted features. Several approaches have been developed for vision-based vegetation detection by using RGB as well as multispectral images, e.g., [54, 57, 98, 177]. Often, popular vegetation indices such as the Normalized Difference Vegetation Index (NDVI) or the Excess Green Index (ExG) are used to separate the vegetation from the background by applying threshold-based approaches. We see a major problem with these kinds of approaches in terms of transferability to other fields, i.e., when the underlying distribution of the indices changes due to different soil types, growth stages, illumination, and weather conditions. Figure 4.1 visually illustrates frequent failure cases: First, a global threshold, here estimated by Otsu's method [123], does not properly separate the vegetation if the plants are in a small growth stage, i.e., when the vegetation is underrepresented. This problem can be solved with adaptive thresholding, by tuning the kernel size and a local threshold for small plants. Second, the adaptive method fails, separating components that should be connected, when using the tuned hyperparameters in another dataset where ei-

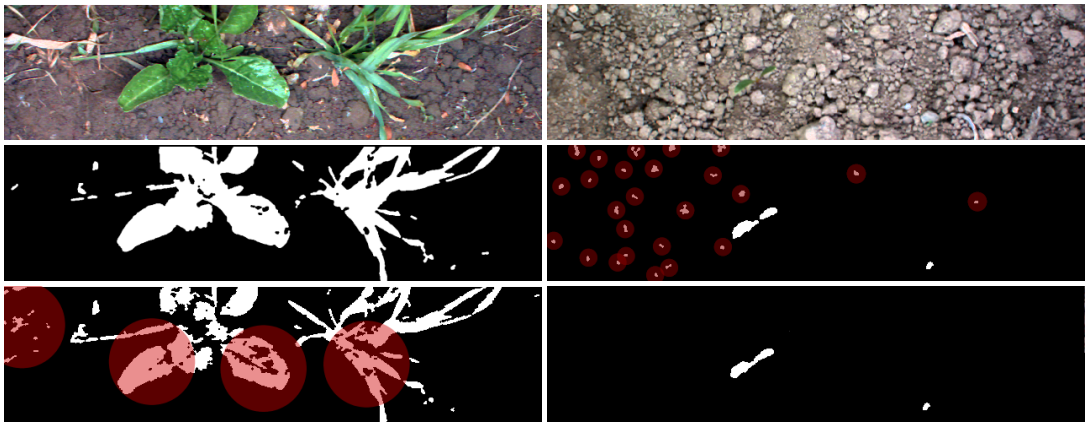

Figure 4.1: Example image of dataset A (left column) and dataset B (right column). Top row: RGB image. Middle row: Vegetation mask obtained by applying a global threshold learned by Otsu's method, which fails for underrepresented vegetation. Bottom row: Vegetation detected by adaptive thresholding, which oversegments big plants. Failure cases are depicted in red.

Figure 4.2: Left: Bonirob system, used for data acquisition and field deployment. Middle: RGB image captured during field test. Right: our corresponding classification result with crops colored green and weeds colored red.

ther the growth stage or the field condition have changed. Thus, we argue that a learning approach that eliminates this pre-segmentation and solves segmentation and classification jointly is needed.

Therefore, in this chapter, we address the problem of classifying standard RGB-only images recorded in the crop fields and identify the weed plants in a pixel-wise manner, without pre-segmentation of the vegetation. As this work illustrates, such a strategy enables us to learn a semantic segmentation which generalizes well over several growth stages as well as field conditions. Furthermore, this approach can operate roughly at the framerate of the camera. This information can, in turn, be used to perform automatic and targeted weed control or to monitor fields and provide a status report to the farmer without human interaction. The main focus is then put into performing RGB-only crop-weed pixel-wise classification (Figure 4.2) without expensive multi-spectral cues, or pre-segmentation. This is done with a special interest in computational efficiency for real-time operation on a mobile robot, as well as a generalization to new, unseen fields.

Figure 4.3: Example images from different datasets. Every dataset contains different crops growth stages, weed types, illumination conditions and soil types. Best viewed in color

4.1 Vision-based Crop vs. Weed Segmentation

In this section, we focus on a new approach to crop-weed classification using *only* RGB data that relies on convolutional neural networks. We aim at feeding additional, task-relevant background knowledge to the network to speed up training and to better generalize to new crop fields. This is a particularly challenging task, as different locations present different weather, soil, and illumination conditions, as depicted in Figure 4.3.

We achieve that by augmenting the input to the CNN with additional channels that have previously been used when designing hand-crafted features for classification [98] and that provide relevant information for the classification process. Our approach yields a pixel-wise semantic segmentation of the image data and can, given the structure of our network, be computed at near the frame-rate of a typical camera.

Our segmentation pipeline is separated in two main steps: First, we compute different vegetation indices and alternate representations that are commonly used in plant classification and support the CNN with that information as additional inputs. This turns out to be specifically useful as the amount of labeled training data for agriculture fields is limited. Second, we employ a self-designed semantic segmentation network to provide a per-pixel semantic labeling of the input data. The following two subsections discuss these two steps of the pipeline in detail.

In sum, we make three key claims, which are the following: Our approach is able to (i) accurately perform pixel-wise semantic segmentation of crops, weeds, and soil, properly dealing with heavily overlapping objects and targeting a large variety of growth stages, without relying on expensive near infra-red information; (ii) act as a robust feature extractor that generalizes well to lighting, soil, and weather conditions not seen in the training set, requiring little data to adapt to the new environment; (iii) work in real-time on a regular GPU such that operation at near the frame-rate of a regular camera becomes possible.

4.1.1 Input Representations

In general, deep learning approaches make as little assumptions as possible about the underlying data distribution and let the optimizer decide how to adjust the parameters of the network by feeding it with enormous amounts of training data. Therefore, most approaches using convolutional neural networks make no modifications to the input data and feed the raw RGB images to the networks.

This means that if we want a deep network to accurately recognize plants and weeds in different fields, we need to train it with a large amount of data from a wide variety of fields, lighting and weather conditions, growth stages, and soil types. Unfortunately, this comes at a high cost and therefore we propose to make assumptions about the inputs to generalize better given the limited training data.

To alleviate this problem, we provide additional vegetation indexes that have been successfully used for the plant classification task and that can be derived from RGB data. In detail, we calculate four different vegetation indices which are often used for the vegetation segmentation [108, 133]: Excess Green (ExG), Excess Red (ExR), Color Index of Vegetation Extraction (CIVE), and Normalized Difference Index (NDI). These indices share the property that they are less sensitive to changes in the mentioned field conditions and therefore can aid the classification task. They are computed straightforwardly as:

$$\mathcal{I}_{\mathrm{ExG}} = 2\,\mathcal{I}_{\mathrm{G}} - \mathcal{I}_{\mathrm{R}} - \mathcal{I}_{\mathrm{B}} \tag{4.1}$$

$$\mathcal{I}_{\mathrm{ExR}} = 1.4\,\mathcal{I}_{\mathrm{R}} - \mathcal{I}_{\mathrm{G}} \tag{4.2}$$

$$\mathcal{I}_{\mathrm{CIVE}} = 0.881\,\mathcal{I}_{\mathrm{G}} - 0.441\,\mathcal{I}_{\mathrm{R}} - 0.385\,\mathcal{I}_{\mathrm{B}} - 18.78745 \tag{4.3}$$

$$\mathcal{I}_{\mathrm{NDI}} = \frac{\mathcal{I}_{\mathrm{G}} - \mathcal{I}_{\mathrm{R}}}{\mathcal{I}_{\mathrm{G}} + \mathcal{I}_{\mathrm{R}}} \tag{4.4}$$

Along with these four additional cues, we use (i) further representations of the raw input such as the HSV color space and (ii) operators on the indices such as the Sobel derivatives, the Laplacian, and the Canny edge detector. All these representations are concatenated to the channel-wise normalized input RGB image and build the input volume which is fed into the convolutional network. In sum, we use the 14 channels listed in Table 4.1, while Figure 4.4 illustrates how some of these representations look like. We show in our experiments that deploying these extra representations to the raw inputs help not only to learn weight parameters which lead to a better generalization property of the network, but also obtain better performance for separating the vegetation, i.e., crops and weeds, from the soil, and speed up the convergence of the training process.

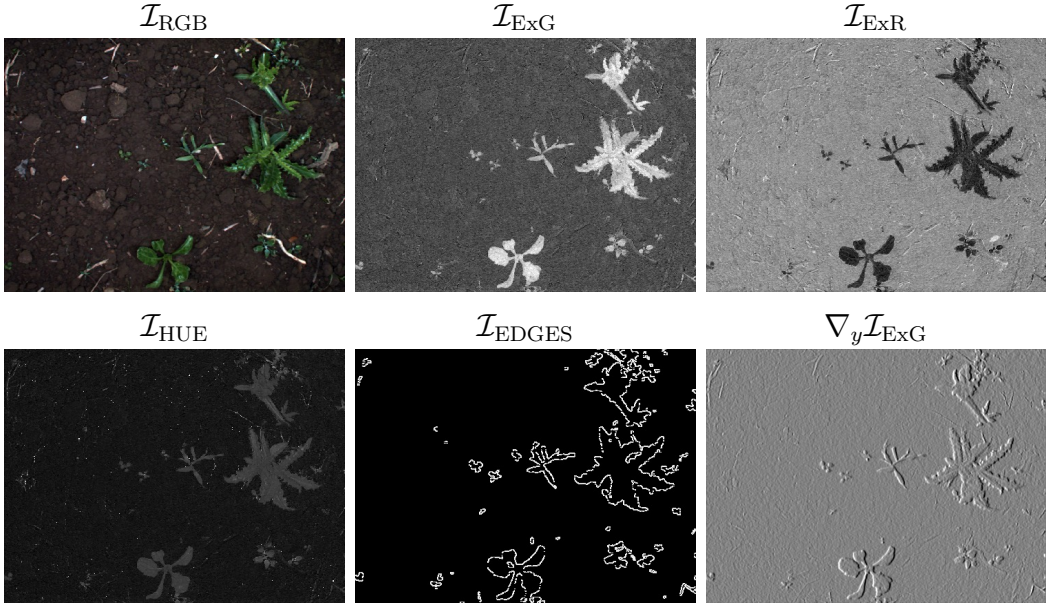

Figure 4.4: Illustration of some of the used alternate representations.

Table 4.1: Indices and representations used as input by our CNN

Input channels for our CNN	
$\mathcal{I}_1$	$\mathcal{I}_{\text{R}}$
$\mathcal{I}_2$	$\mathcal{I}_{\text{G}}$
$\mathcal{I}_3$	$\mathcal{I}_{\text{B}}$
$\mathcal{I}_4$	$\mathcal{I}_{\text{ExG}}$
$\mathcal{I}_5$	$\mathcal{I}_{\text{ExR}}$
$\mathcal{I}_6$	$\mathcal{I}_{\text{CIVE}}$
$\mathcal{I}_7$	$\mathcal{I}_{\text{NDI}}$
$\mathcal{I}_8$	$\mathcal{I}_{\text{HUE}}$ (from HSV colorspace)
$\mathcal{I}_9$	$\mathcal{I}_{\text{SAT}}$ (from HSV colorspace)
$\mathcal{I}_{10}$	$\mathcal{I}_{\text{VAL}}$ (from HSV colorspace)
$\mathcal{I}_{11}$	$\nabla_x \mathcal{I}_{\text{ExG}}$ (Sobel in x direction on $\mathcal{I}_{\text{ExG}}$)
$\mathcal{I}_{12}$	$\nabla_y \mathcal{I}_{\text{ExG}}$ (Sobel in y direction on $\mathcal{I}_{\text{ExG}}$)
$\mathcal{I}_{13}$	$\nabla^2 \mathcal{I}_{\text{ExG}}$ (Laplacian on $\mathcal{I}_{\text{ExG}}$)
$\mathcal{I}_{14}$	$\mathcal{I}_{\text{EDGES}}$ (Canny Edge Detector on $\mathcal{I}_{\text{ExG}}$)

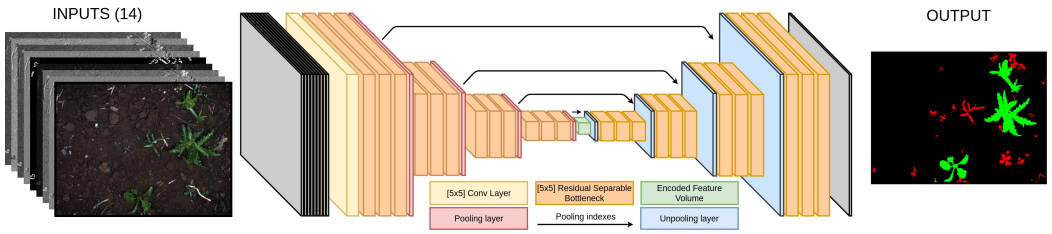

Figure 4.5: Detailed encoder-decoder architecture used for the pixel-wise semantic segmentation. Best viewed in color.

4.1.2 Network Architecture

Semantic segmentation is a memory and computationally expensive task. State-of-the-art algorithms for multi-class pixel-wise classification use CNNs that have tens of millions of parameters and therefore need massive amounts of labeled data for training. Another problem with these networks is the fact that they often cannot be run fast enough for our application. Current state-of-the-art CNNs for this task [23, 195] processes around 2 images per second, whereas we target on a classification rate of at least 10 Hz.

Since our specific task of crop vs. weed segmentation has a much narrower target space compared to the general architectures designed for hundreds or even thousands of classes, we can design an architecture that meets the targeted speed and efficiency. We propose an end-to-end encoder-decoder semantic segmentation network, see Figure 4.5, that can accurately perform the pixel-wise prediction task while running at 20+ Hz, can be trained end-to-end with a moderate size training dataset, and has less than 30 000 parameters. We design the architecture taking some design cues from Segnet [7] and Enet [126] into account and adapt them to the task at hand. Our network is based on the following building blocks:

Input: We use the representation volume described in Table 4.1 as the input to our network. Before it is passed to the first convolutional layer, we perform a resizing to 512×384 pixels and channel-wise contrast normalization.

Convolutional Layer: We define our convolutional layers as a composite function of a convolution followed by batch normalization and use the rectified linear unit (ReLU) for the non-linearity. The batch normalization operation prevents an internal covariate shift and allows for higher learning rates and better generalization capabilities. The ReLU is computationally efficient and suitable for training deep networks. All convolutional layers use zero-padding to avoid washing out edge information.

Residual Separable Bottleneck: To achieve a faster processing time while keeping the receptive field, we propose to use the principal building block for our network, which is built upon the ideas of (i) residual connections, (ii) bottlenecks,

and (iii) separating the convolutional operations.

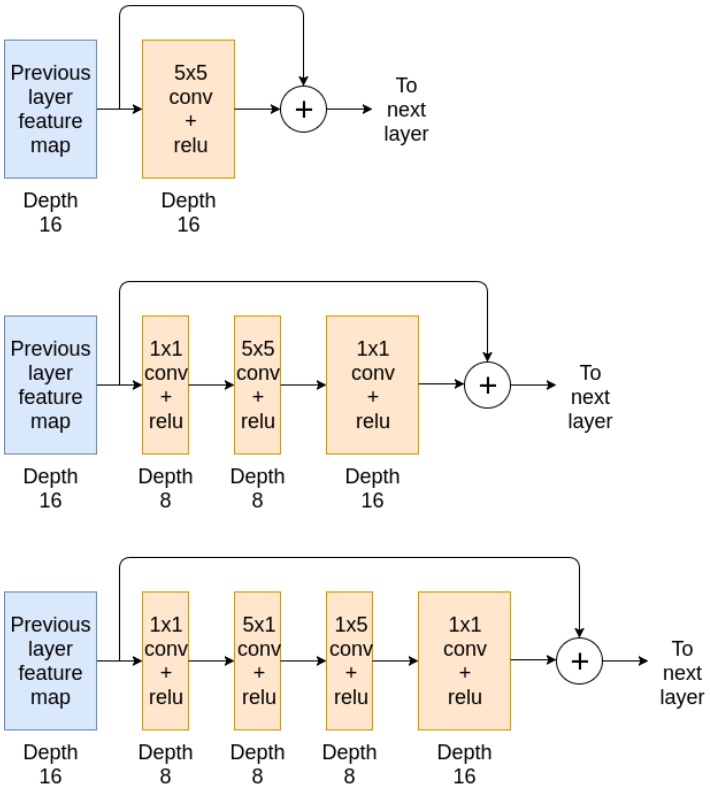

Figure 4.6: Building the residual block to replace a $[5 \times 5]$ layer.

Figure 4.6 illustrates the evolution from a conventional $[5 \times 5]$ convolutional layer to a residual separable bottleneck. The top row of Figure 4.6 shows the addition of a residual connection, which adds the input of the convolution to its result. The addition of this residual connection helps with the degradation problem that appears when training very deep networks which makes them obtain a higher training error than their shallower counterparts [60].

The middle row of Figure 4.6 adds $[1 \times 1]$ convolutions that reduce the depth of the input volume so that the expensive $[5 \times 5]$ operation does not need to run in the whole depth. For this, we use $8 \times [1 \times 1]$ kernels, which halve the depth of the input volume, and $16 \times [1 \times 1]$ kernels to expand the result, which needs to match the depth of the input to be added to it. This reduces the number of calculations per operation of the kernel from 6 400 FLOPs to 1 856 FLOPs.

Finally, the bottom row of Figure 4.6 shows the separation of each $[5 \times 5]$ convolution into a $[5 \times 1]$ convolution followed by a $[1 \times 5]$ convolution. This further reduces the operations of running the module in a $[5 \times 5]$ window from 1 856 FLOPs to 896 FLOPs.

These design choices also decrease the number of parameters for the equivalent layer to the $[5 \times 5]$ convolution with 16 kernels from 6 400 to 896 parameters.

Un-pooling with Shared Indexes: The un-pooling operations in the decoder are performed sharing the pooling indexes of the symmetrical pooling operation in the encoder part of the network. This allows the network to maintain the information about the spatial positions of the maximum activations on the encoder part without the need for transposed convolutions, which are comparably expensive to execute. Therefore, after each un-pooling operation, we obtain a sparse feature map, which the subsequent convolutional layers learn how to densify. All pooling layers are $[2 \times 2]$ with stride 2.

Output: The last layer is a linear operation followed by a softmax activation predicting a pseudo-probability vector of length 3 per pixel, where each element represents the probability of the pixel belonging to the class background, weed, or crop.

We use these building blocks to create the network depicted in Figure 4.5, which consists of an encoder-decoder architecture. The encoder part has $13 \times [5 \times 5]$ convolutional layers containing 16 kernels each and 4 pooling layers that reduce the input representation into a small code feature volume. This is followed by a decoder with 12 convolutional layers containing 16 kernels each and 4 un-pooling layers that upsample this code into a semantic map of the same size of the input. The design of the architecture is inspired by Segnet's design simplicity, but to make it smaller, easier to train, and more efficient, we replace 24 of the 25 convolutional layers by our proposed residual separable convolutional bottlenecks. The 5×5 receptive fields of the convolutional layers, along with the pooling layers add up to an equivalent receptive field in the input image plane of 200×200 pixels. This is sufficient for our application since it is a bigger window than the biggest plant we expect in our data. The proposed number of layers, kernels per layer, and reduction factor for each bottleneck module was chosen by training several networks with different configurations, and reducing its size until we reached the smallest configuration that did not result in a considerable performance decrease.

4.2 Experimental Evaluation

The experiments are designed to evaluate the accuracy and efficiency of our method and to support the claims made in the introduction that our classifier is able to perform accurate pixel-wise classification of value crops and weeds, generalize well, and run in real-time. We implemented the whole approach presented in this section relying on the Google TensorFlow library and OpenCV to compute alternate representations of the inputs. We tested our results using a Bosch Deepfield Robotics BoniRob UGV.

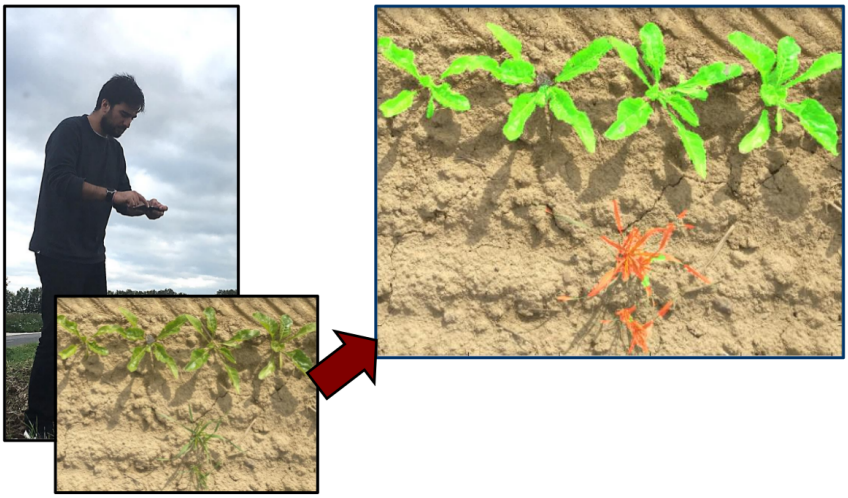

Figure 4.7: Removing the reliance on NIR cues allows images captured by any RGB camera to be processed by an algorithm, even the ones from a mobile phone.

4.2.1 Training and Testing Data

To evaluate the performance of the network, we use three different datasets captured in Bonn, Germany; Stuttgart, Germany; and Zurich, Switzerland, see Table 4.2. Part of the data from Bonn is publicly available [22]. All datasets contain plants and weeds in all growth stages, with different soil, weather, and illumination conditions. Figure 4.3 illustrates the variance of the mentioned conditions for each dataset. The visual data was recorded with the 4-channel RGB+NIR camera JAI AD-130 GE mounted in nadir view. For our approach, we use solely the RGB channels because we aim to perform well with an off-the-shelf RGB camera, see Figure 4.7, but the additional NIR information allows us to compare the performance with an approach that exploits the additional NIR information.

Table 4.2: Dataset information

	Bonn	**Zurich**	**Stuttgart**
# images	10 036	2 577	2 584
# crops	27 652	3 983	10 045
crop pixels	1.7%	0.4%	1.5%
# weeds	65 132	14 820	7 026
weed pixels	0.7%	0.1%	0.7%

To show the generalization capabilities of the pipeline, we train our network using only images from Bonn, captured over the course of one month, and containing images of several growth stages. We separate the dataset in 70%-15%-15% for training, validation, and testing, and we report our results only in the latter,

Table 4.3: Pixel-wise test sets performance. Trained in 70% Bonn, reporting 15% held-out Bonn, 100% Stuttgart, and 100% Zurich.

Dataset	Network	mIoU[%]	IoU[%]			Precision[%]			Recall[%]		
			Soil	Weeds	Crops	Soil	Weeds	Crops	Soil	Weeds	Crops
Bonn	$\mathcal{I}_{\text{RGB}}$	59.98	99.08	20.64	60.22	99.92	28.97	66.49	99.15	41.79	82.45
	$\mathcal{I}_{\text{RGB}}$+$\mathcal{I}_{\text{NIR}}$	76.92	99.29	49.42	82.06	99.88	52.90	84.19	99.33	**88.24**	97.01
	$\mathcal{I}_1$...$\mathcal{I}_{14}$ (ours)	**80.8**	**99.48**	**59.17**	**83.72**	**99.95**	**65.92**	**85.71**	**99.53**	85.25	**97.29**
Zurich	$\mathcal{I}_{\text{RGB}}$	38.25	96.84	14.26	3.62	96.95	14.96	3.78	96.88	35.35	45.86
	$\mathcal{I}_{\text{RGB}}$+$\mathcal{I}_{\text{NIR}}$	41.23	98.44	16.83	8.43	99.68	19.03	9.07	98.46	**51.27**	54.46
	$\mathcal{I}_1$...$\mathcal{I}_{14}$ (ours)	**48.36**	**99.27**	**23.40**	**22.39**	**99.90**	**31.43**	**23.05**	**99.36**	47.79	**88.74**
Stuttgart	$\mathcal{I}_{\text{RGB}}$	48.09	99.18	21.40	23.69	99.84	21.90	52.43	99.34	42.95	28.95
	$\mathcal{I}_{\text{RGB}}$+$\mathcal{I}_{\text{NIR}}$	55.82	98.54	23.13	45.80	99.85	25.28	68.76	98.69	**49.10**	57.84
	$\mathcal{I}_1$...$\mathcal{I}_{14}$ (ours)	**61.12**	**99.32**	**26.36**	**57.65**	**99.86**	**37.58**	**68.77**	**99.45**	46.90	**78.09**

as well as the whole of the two other datasets from Zurich and Stuttgart, some even recorded in different years. We train the network using the 70% part of the Bonn dataset extracted for that purpose and perturb the input data by performing random rotations, scalings, shears, and stretches. We use stochastic gradient descent with a batch size of 15 for each step, which is the maximum we can fit in a single GPU. Furthermore, we use a weighted cross-entropy loss, to handle the imbalanced number of pixels of each class, due to the soil dominance, and the Adam optimizer for the calculation of the gradient steps. Training for 200 epochs takes roughly 48 hours on an NVIDIA GTX1080Ti.

To show the effect in performance obtained by the extra channels, we train three networks using different types of inputs: one based solely on RGB images, another one based on RGB and extra representations, and finally the reference baseline network using RGB+NIR image data.

4.2.2 Performance of the Semantic Segmentation

This experiment is designed to support our first claim, which states that our approach can accurately perform pixel-wise semantic segmentation of crops, weeds, and soil, properly dealing with heavy plant overlap in all growth stages.

In Table 4.3, we show the pixel-wise performance of the classifier tested in the 15% held out test set from Bonn, as well as the whole datasets from Zurich and Stuttgart. These results show that the network trained using the RGB images in conjunction with the extra computed representations of the inputs outperforms the network using solely RGB images significantly in all categories, performing comparably with the network that uses the extra visual cue from the NIR information, which comes with a high additional cost as specific sensors have to be employed.

Even though the pixel-wise performance of the classifier is important, in order to perform automated weeding it is important to have an object-wise metric for the classifier's performance. We show this metric in Table 4.4, where we

Table 4.4: Object-wise test performance. Trained in 70% Bonn, reporting 15% held-out Bonn, 100% Stuttgart, and 100% Zurich.

Dataset	Network	mAcc[%]	Precision[%]		Recall[%]	
			Weeds	Crops	Weeds	Crops
Bonn	$\mathcal{I}_{\text{RGB}}$	86.34	83.63	81.14	91.99	80.42
	$\mathcal{I}_{\text{RGB}}$+$\mathcal{I}_{\text{NIR}}$	93.72	90.51	**95.09**	**94.79**	89.46
	$\mathcal{I}_1$...$\mathcal{I}_{14}$ (ours)	**94.74**	**98.16**	91.97	93.35	**95.17**
Zurich	$\mathcal{I}_{\text{RGB}}$	45.51	59.75	19.71	42.52	23.66
	$\mathcal{I}_{\text{RGB}}$+$\mathcal{I}_{\text{NIR}}$	68.03	67.41	46.78	**65.31**	49.32
	$\mathcal{I}_1$...$\mathcal{I}_{14}$ (ours)	**72.08**	**67.91**	**72.55**	63.33	**64.94**
Stuttgart	$\mathcal{I}_{\text{RGB}}$	46.05	42.32	42.03	46.1	25.01
	$\mathcal{I}_{\text{RGB}}$+$\mathcal{I}_{\text{NIR}}$	73.99	74.30	**70.23**	**71.35**	53.88
	$\mathcal{I}_1$...$\mathcal{I}_{14}$ (ours)	**76.54**	**87.87**	65.25	64.66	**85.15**

analyze all objects with an area bigger than 50 pixels. This number is calculated by dividing our desired minimum object detection size of $1\,\text{cm}^2$ by the spatial resolution of $2\,\text{mm}^2/\text{px}$ in our 512×384 resized images. We can see that in terms of object-wise performance the network using all representations outperforms its RGB counterpart. Most importantly, in the case of generalization to the datasets of Zurich and Stuttgart, this difference becomes critical, since the RGB network yields a performance so low that it renders the classifier unusable for most tasks.

Note that the network using RGB and extra representations is around 30% faster to converge to 95% of the final accuracy than its RGB counterpart, and roughly 15% faster than the network using the NIR channel.

4.2.3 Labeling Cost for Adaptation to New Fields

This experiment supports our second claim, which states that our approach can act as a robust feature extractor for images in conditions not seen in the training set, requiring little data to adapt to the new environment.

One way to analyze the generalization performance of the approach is to analyze the amount of data that needs to be labeled in a new field for the classifier to achieve state-of-the-art performance. For this, we separate the Zurich and Stuttgart datasets in halves, and we keep 50% of it for testing. From each of the remaining 50%, we extract sets of 10, 20, 50, and 100 images, and we retrain the last layer of the network trained in Bonn, using the convolutional layers in the encoder and the decoder as a feature extractor. We further separate this small sub-samples in 80%-20% for training and validation and we train until convergence, using early stopping, which means that we stop training when the validation error starts to increase. This is to provide an automated approach to the re-training so that it can be done without the supervision of an expert, since shipping a graduate student with each robot is not a feasible enterprise.

Table 4.5: Object-wise test performance retraining in N images of Zurich dataset.

Inputs	Nr. Images	mAcc[%]	Precision[%]		Recall[%]	
			Weeds	Crops	Weeds	Crops
$\mathcal{I}_{\text{RGB}}$	10	60.08	58.75	62.22	73.03	57.99
	20	71.38	61.38	81.42	76.72	64.58
	50	74.08	63.00	85.42	74.14	68.34
	100	82.26	65.50	85.59	74.97	69.91
$\mathcal{I}_1 ... \mathcal{I}_{14}$ (ours)	10	83.90	69.23	80.73	71.71	76.18
	20	85.31	75.85	79.12	67.67	84.01
	50	86.25	76.50	85.24	71.55	84.33
	100	**89.55**	**85.89**	**89.52**	**89.69**	**86.76**

Table 4.6: Object-wise test performance retraining in N images of Stuttgart dataset.

Inputs	Nr. Images	mAcc[%]	Precision[%]		Recall[%]	
			Weeds	Crops	Weeds	Crops
$\mathcal{I}_{\text{RGB}}$	10	71.76	93.88	52.59	56.57	81.10
	20	72.30	94.40	57.72	51.43	86.35
	50	72.97	94.33	59.70	54.11	87.69
	100	73.34	95.20	63.26	56.36	87.66
$\mathcal{I}_1 ... \mathcal{I}_{14}$ (ours)	10	81.40	89.48	64.42	78.74	79.69
	20	81.84	87.83	68.18	81.45	74.61
	50	86.75	94.68	71.34	83.22	90.23
	100	**91.88**	**95.45**	**86.58**	**89.08**	**91.43**

We show the results of the retraining on the Zurich dataset in Table 4.5 and Figure 4.8, where we can see that the performance of the RGB network when relabeling 100 images is roughly the same as the one using all input representations when the latter is using only 10 images, thus significantly reducing the relabeling effort. We also show that we can get values of precision and recall in the order of 90% when using 100 images for the relabeling in the case of our network, which exploits additional channels. In Table 4.6 and Figure 4.8, we show the same for the Stuttgart dataset, but in this case, the RGB network fails to reach an acceptable performance, while the accuracy of our approach grows linearly with the number of images used.

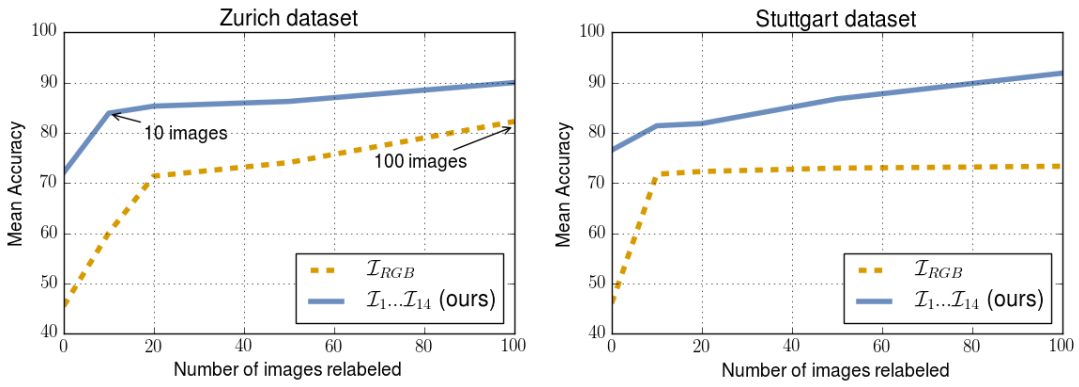

Figure 4.8: Object-wise mean accuracy vs. relabeling effort. Zero images means no retraining.

4.2.4 Runtime

This experiment supports our third claim, which states that our approach can be run in real-time, and therefore it can be used for online operation in the field, running in hardware that can be fitted in mobile robots.

We show in Table 4.7 that even though the architecture using all extra representations of the RGB inputs has a speed penalty, due to the efficiency of the network we can run the full classifier at more than 20 frames per second on the hardware in our UGV, which consists of an Intel i7 CPU and an NVIDIA GTX1080Ti GPU. Furthermore, we tested our approach in the Jetson TX2 platform, which has a very small footprint and only takes 15 W of peak power, making it suitable for operation on a flying vehicle, and here we still obtain a frame rate of almost 15 Hz.

Table 4.7: Runtime in different devices.

Inputs	FLOPS	Hardware	Preproc.	Network	Total	FPS
RGB	1.8G	i7+GTX1080Ti	-	31ms	31ms	32.2
		Tegra TX2 SoC	-	65ms	65ms	15.2
All	2G	i7+GTX1080Ti	6ms	38ms	44ms	22.7
		Tegra TX2 SoC	6ms	62ms	68ms	14.7

4.3 Related Work

In the context of crop classification, several supervised learning algorithms have been proposed in the past few years [58, 101, 98, 105, 110, 116, 137]. McCool *et al.* [105] address the crop and weed segmentation problem by using an ensemble of small CNN's compressed from a more complex pre-trained model and report an accuracy of nearly 94%. Haug *et al.* [58] propose a method to distinguish carrot plants and weeds in RGB and near infra-red images. They obtain an average accuracy of 94% on an evaluation dataset of 70 images. Mortensen *et al.* [116] apply a deep CNN for classifying different types of crops to estimate their amounts of biomass. They use RGB images of field plots captured at 3 m above ground and report an overall accuracy of 80% evaluated on a per-pixel basis. Potena *et al.* [137] present a multi-step visual system based on RGB+NIR imagery for crop and weed classification using two different CNN architectures. A shallow network performs the vegetation detection and then a deeper network further distinguishes the detected vegetation into crops and weeds. They perform a pixel-wise classification followed by a voting scheme to obtain predictions for connected components in the vegetation mask. They report an average precision of 98% in case the visual appearance has not changed between training and testing.

In previous works [101, 98], our group presented a vision-based classification system based on RGB+NIR images for sugar beets and weeds that relies on NDVI-based pre-segmentation of the vegetation. The approach combines some appearance and geometric properties using a random forest classifier and obtains classification accuracies of up to 96% on a pixel level. The works, however, also show that the performance decreases to an unsuitable level for weed control applications when the appearance of the plants changes substantially. In Chapter 3, we presented a 2-step approach that pre-segments vegetation objects by using the NDVI index and afterward uses a CNN classifier on the segmented objects to distinguish them into crops and weeds. We showed that we can obtain state-of-the-art object-wise results in the order of 97% precision, but only for early growth stages, since the approach is not able to deal with overlapping plants due to the pre-segmentation step. However, that approach can do the classification step with no expert knowledge, unlike this work.

Lottes *et al.* [103] address the generalization problem by using the crop arrangement information, for example from seeding, as prior and exploit a semi-supervised random forest-based approach that combines this geometric information with a visual classifier to quickly adapt to new fields. Hall *et al.* [56] also address the issue of changing feature distributions. They evaluate different features for leaf classification by simulating real-world conditions using basic data augmentation techniques. They compare the obtained performance by selectively using different handcrafted and CNN features and conclude that CNN features can support the robustness and generality of a classifier.

In this chapter, we also explore a solution to this problem using a CNN-based feature extractor and classifier for RGB images that uses no geometric prior, and generalizes well to different soil, weather, and illumination conditions. To achieve this, we use a combination of end-to-end efficient semantic segmentation networks and several vegetation indices and preprocessing mappings to the RGB images.

A further challenge for these supervised classification approaches is the necessary amount of labeled data required for retraining to adapt to a new field. Some approaches have been investigated to address to this problem [35, 142, 183]. Wendel and Underwood [183] address this issue by proposing a method for training data generation. They use a multi-spectral line scanner mounted on a field robot and perform a vegetation segmentation followed by a crop-row detection. Subsequently, they assign the label crop for the pixels corresponding to the crop row and the remaining ones as weeds. Rainville *et al.* [142] propose a vision-based method to learn a probability distribution of morphological features based on a previously computed crop row. Di Cicco *et al.* [35] try to minimize the labeling effort by constructing synthetic datasets using a physical model of a sugar beet leaf. Unlike most of these approaches, our purely visual approach relies solely on

RGB images and can be applied to fields where there is no crop row structure, as our experimental evaluation showed.

4.4 Conclusion

In this chapter, we presented an approach to pixel-wise semantic segmentation of crop fields identifying crops, weeds, and background in real-time solely from RGB data. We proposed a deep encoder-decoder CNN for semantic segmentation that is fed with a 14-channel image storing vegetation indexes and other information that in the past has been used to solve crop-weed classification tasks. By feeding this additional, task-relevant background knowledge to the network, we can speed up training and improve the generalization capabilities on new crop fields, especially if the amount of training data is limited. We implemented and thoroughly evaluated our system on a real agricultural robot operating using data from three different places in Germany and Switzerland. Our results suggest that our system generalizes well, can operate at around 15 FPS in embedded edge devices, and is suitable for online operation in the fields.

Chapter 5

Instance Segmentation in Urban Environments

As we have mentioned multiple times through this thesis already, perception is one of the main building blocks for robotic applications that need interaction with real-world, dynamic environments such as agricultural fields, city roads, or even domestic environments. An accurate understanding of what is present and happening in the scene is absolutely key for robots to operate safely and in a situation-dependent manner. Semantic information about the environment in the form of object detection, semantic segmentation, and instance segmentation have been exploited for many robotics tasks such as mapping [37, 75, 106, 168], visual place recognition [45], manipulation [15, 161], and agriculture [98, 111], Chapter 3, and Chapter 4.

One of four different types of approaches is typically used in this context: semantic segmentation, object detection, instance or panoptic segmentation. *Semantic segmentation* (see Figure 5.1, middle) conveniently provides a class label *for each pixel* of an image, and therefore enables applications that require accurate object masks such as removal of dynamic objects for visual odometry. However, it does not provide an association of the pixels to an object instance. This is inconvenient for robotic tasks where the location of each object is of interest, for example, to be used as a visual landmark, for object manipulation or collision avoidance. Furthermore, performing semantic segmentation is often computationally demanding. This is especially true if using high-resolution images, which is often desired for high accuracy and handling objects at a far distance. On the other side, *object detection* provides a class label for each object instance jointly with the regression of the image coordinates of a bounding box that encloses the object. This approach is popular as it has the advantage of fast inference. Unfortunately, object detection does not provide an object mask, and the correlation between the object boundaries and the bounding boxes is not trivial.

Figure 5.1: Desired workflow. Top: Input RGB image. Middle: Semantic segmentation mask. Bottom: Instance segmentation mask.

The key sub-tasks for semantic scene understanding in robotics are *instance segmentation* (see Figure 5.1, bottom) and *panoptic segmentation* (which combines semantic and instance segmentation as a single task, with a unified metric). Both these tasks provide labels for each object instance, but also provide an accurate segmentation mask and thus enables a further, task-specific, analysis of the image content. A typical example in robotics of the type of application that this enables is the accurate removal of dynamics from the scene to be able to do simultaneous localization and mapping under the usually required static-world assumption. This is, of course, also possible with *semantic segmentation* labels, but these make obstacle avoidance and obstacle intent prediction harder, or even impossible, in the usual case that objects in the scene overlap in the camera view. Furthermore, *panoptic segmentation* provides class masks for non-object classes as well. We focus on the latter in Chapter 8, and instead, on this chapter we focus solely on the former.

The main contribution represented in this chapter is a novel approach that performs *instance and semantic segmentation* of the object classes in the scene, both efficiently and effectively for driving platforms, which need quick reaction times for safe actuation. We propose a convolutional neural network architecture that uses superpixel summarization of locally connected regions of an image, and combines object detection and a metric learning pipeline to speed up joint semantic and instance segmentation without sacrificing accuracy.

Our CNN predicts, for each output superpixel, the probability of it being an instance center, a high-dimensional embedding from a metric learning loss, and a softmax probability for the semantic segmentation task. This approach allows us to perform fast upsampling without sacrificing mask accuracy, while at the same time speeding up the clustering of the embeddings to obtain an individual instance mask for each object in the scene. Along with Chapter 4, this approach is another example of how domain-specific priors can be included to improve some of the aspects of the perception pipeline (such as performance or runtime). As the reader can notice, this is a recurrent topic throughout this thesis, since as we approach the limits of what can be done with end-to-end, purely data-driven, deep learning algorithms, these priors are what make the whole design paradigm work in the real world.

5.1 Semantic Segmentation using Neighborhood Information

The main goal of our work is to obtain an accurate instance segmentation masks of objects from RGB camera images in a timely manner. This enables online decision making based on the extracted semantics, for tasks such as semantic landmark extraction, obstacle avoidance, manipulation, intent prediction, path planning, or visual odometry. As each of these tasks needs a different balance between the localization of each object and the accuracy of their masks, it is important to have algorithms that can perform both tasks accurately, while fast enough to process the camera images at frame-rate speeds. We propose a CNN-based algorithm that combines detection and segmentation with superpixel summarization, allowing us to obtain accurate semantic and instance labels for each pixel of an image. As our approach operates in a low dimensional feature-space grid, it is fast to run without sacrificing mask or instance performance.

Our method uses a common CNN encoder extracting features from RGB images at different resolutions, and has three separate decoder heads, see Figure 5.2. Each decoder head combines and upsamples the multi-resolution features into a low-dimensional grid, which makes the processing fast. We explain these three output grids extensively, and we call the value at a certain (x, y) position of this grid a "grid element". Such an element addresses all corresponding feature values at that spatial position. The three heads predict: (i) a semantic segmentation mask which maps each grid element to a softmax pseudo-probability distribution over the desired semantic classes; (ii) a high-dimensional embedding for each grid element, which is to be close in Euclidean similarity for elements belonging to the same instance, and distant otherwise; and (iii) the confidence of each grid element

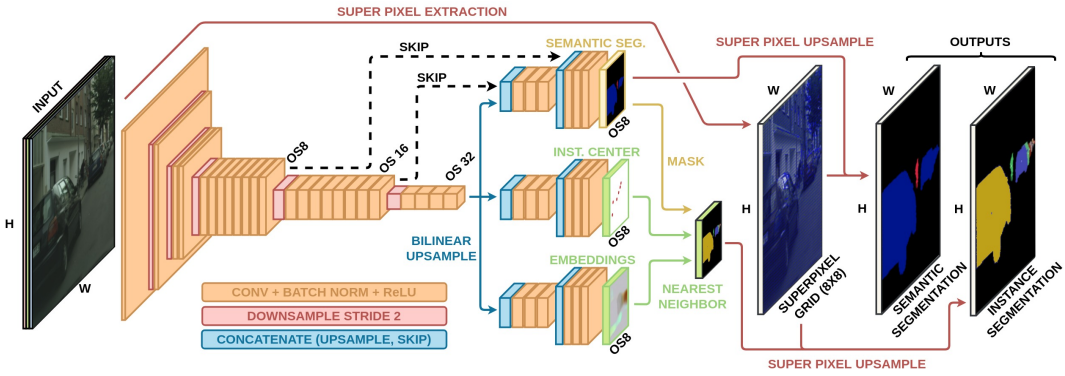

Figure 5.2: Our architecture. The different resolution features are skipped to all decoders in their respective output stride (OS). Each decoder predicts a volume of lower resolution than the input, in this case of OS8, reducing the number of pixels to cluster by a factor of 64. After the decoders predict the semantic mask and the centers and embeddings are joined into individual instances, the superpixels are used to upsample the results. The encoder shown is Darknet53 [144].

being an object center. Parallelly, the input is processed by a fast GPU-based superpixel algorithm [147] based on SLIC [2], which is able to upsample each "grid element" into the locally connected pixels in the original resolution output.

Our approach can be summarized in three main steps. First, a CNN backbone summarizes the image as a set of features of different resolution, and three task decoders upsample these features into task grids of lower resolution than the input. These decoders contain the semantic segmentation, center confidences, and embeddings respectively for each of the groups of input pixels that are mapped to it. Second, and in parallel with the CNN, a fast superpixel extraction [147] of the original image is performed resulting in a mapping used to upsample the decoder grids. Finally, post-processing is performed to map each embedding to an individual object center and extract the instances, previous to upsampling using the mappings from step two.

5.1.1 Joint Semantic and Instance Segmentation CNN

Our CNN structure is composed of four main components: (i) a fully convolutional encoder which extracts features at different resolutions for the decoders, (ii) a decoder which infers a downsampled semantic segmentation softmax distribution over the semantic classes, (iii) a decoder which infers the confidence of each superpixel being an object center, and (iv) a decoder which infers a 32-dimensional embedding for each superpixel using a discriminative metric loss. All four components are trained jointly using a weighted sum of the three task losses. In the following subsections, we introduce each of these modules in detail.

5.1.1.1 Encoders

We use two different convolutional backbone architectures for the multi-resolution feature extraction with different levels of descriptiveness, defined by their size and the number of layers. On the computationally expensive side of the spectrum, we use Darknet 53 [144] (DN53), which is a ResNet [60] inspired architecture and has proven to work well for the object detection task as a part of the YOLOv3 architecture. This architecture also obtains top-1 accuracy on the validations set of ImageNet-1K [34] of 77.2%, which is the current state of the art, and higher than ResNet101, which is 50% slower to run. This makes it a well-suited feature extractor for our backbone. On the other side of the descriptiveness spectrum, we use a MobilenetsV2 [159] (MNv2) encoder, which has an order of magnitude less parameters than Darknet 53 and is designed to maximize efficiency for running on mobile devices. This backbone achieves a top-1 ImageNet accuracy of 72%. This is lower than the more expensive Darknet 53 but is the current state of the art for real-time applications. Since the publication of this work, newer mobile-oriented models have been released [64], but because they were searched through neural architecture search to run fast on mobile CPUs, their GPU performance is poor, so we stand with our decision to use MobilenetsV2. Before attaching the decoders, we pre-train both backbones on ImageNet to accuracy, using the standard output stride (OS) of both backbones of 32, which means that the last layer will be downsampled 32 times from the original image size. Our framework allows the extraction of any feature resolution of the model to skip to the decoder, and to modify the output stride of the final layer by using dilated convolutions [23], to be able to segment small objects at the expense of extra calculations. We use two different encoders to show that the gains of our approach are similar regardless of the architecture used, meaning that once a better feature extractor is designed, our method can be used on top of it.

5.1.1.2 Semantic Segmentation Decoder

All three decoders have the same architecture, but their last layer is passed through a different activation function. Furthermore, during training, they are optimized with different losses. On top of the encoder, we attach a module that upsamples the last layer's features and concatenates them to their matching skip resolution in the encoder (see Figure 5.2). After the concatenation we use a $[1 \times 1]$ convolution that squashes the upsampled features with the skipped ones, and 2 layers of $[3 \times 3]$ convolutions to combine them and learn task-specific parameters. This is done iteratively until the output volume matches the size of the grid needed to perform the superpixel upsampling. For the semantic segmentation head, the output depth is the number of classes, and the activation function is

Figure 5.3: Instance labels and their corresponding 32-dimensional embeddings randomly projected to 3 dimensions to show them in RGB. Best viewed in color.

a softmax $\hat{y}_c = \frac{e^{\text{logit}_c}}{\sum_c e^{\text{logit}_c}}$, where logit_c is the unbounded output in the slice corresponding to class c. This gives a pseudo-probability distribution per grid-element, which is optimized using a weighted cross-entropy loss:

$$L_{\text{semantic}} = -\sum_{c=1}^{C} w_c \; y_c \; \log\left(\hat{y}_c\right), \tag{5.1}$$

$$w_c = \frac{1}{\log\left(f_c + \epsilon\right)}, \qquad f_c = \frac{1}{P}\sum_{p=1}^{P} \begin{cases} 1 & \text{if } p = c \\ 0 & \text{if } p \neq c \end{cases}, \tag{5.2}$$

where w_c which penalizes class c according to the inverse of its frequency in the ground truth, bounded by a parameter ϵ which is set to 1.02 in all our experiments.

5.1.1.3 Embedding Decoder

The objective of this branch is to provide a high-dimensional embedding (with 32 dimensions in our case) that has a low Euclidean distance to all other grid elements of the same instance, and a high Euclidean distance to all other instances. Figure 5.3 shows this in action by randomly projecting all 32-dimensional embeddings into 3D space so that each embedding can be mapped to an RGB value (for illustrative reasons). The embedding branch is analog to the semantic segmentation branch, in terms of upsampling, concatenating, squashing, and adding extra convolutional layers, but the main difference lies in the lack of an activation function. For this layer, instead of a softmax activation, we use the unbounded logits, which we call $\hat{e}$. Following [16], we define three hinged losses to achieve this purpose. In all equations, K is the number of instances, P_k is the number of grid elements in instance k, $\|\cdot\|$ is the L2-norm, and $[x]^+$ means the positive part of x, meaning $\max(0, x)$, which hinges the loss. The first loss is the attraction loss

$$L_{\text{attract}} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{P_k}\sum_{p=1}^{P_k}\left[\,\|\hat{e}_{k,c} - \hat{e}_{k,p}\| - \delta_{\text{a}}\right]^+, \tag{5.3}$$

which defines that all pixels of the same instance should have a low Euclidean distance to the embedding in its center. Following De Brabandere [16], this loss is hinged, which means that the embeddings that are already "close enough" to the center embedding do not receive a loss. This allows the embeddings to move around improving training and inference stability. This is modulated by the parameter δ_a, set to 0.1 for all our experiments. Note that in [16], the distance is calculated to the mean of the embedding due to the lack of the concept of object center. The second loss is the repelling loss:

$$L_{\text{repel}} = \frac{1}{K(K-1)} \sum_{\substack{k_A=1 \\ k_A \neq k_B}}^{K} \sum_{k_B=1}^{K} \left[\delta_{\text{r}} - \| \hat{e}_{k_A,c} - \hat{e}_{k_B,c} \| \right]^{+} \tag{5.4}$$

This loss pushes the object centers from different instances away from each other in embedding space. Analogously to the attraction loss, this loss is hinged to allow the embeddings to move around when they are "far enough". This is modulated by δ_r, which is set to 1.0. The third loss

$$L_{\text{reg}} \;=\; \frac{1}{K} \sum_{k=1}^{K} \| \hat{e}_{k,c} \|, \tag{5.5}$$

penalizes the norm of all embeddings in the object centers to improve stability, making the embeddings stay close to the origin, and has no further meaning for the approach. We combine the three loss functions as a weighted sum and use it to backpropagate through the embedding decoder, as well as the encoder as:

$$L_{\text{embed}} = \alpha \; L_{\text{attract}} + \beta \; L_{\text{repel}} + \gamma \; L_{\text{reg}}, \tag{5.6}$$

with $\alpha = \beta = 1.0$ and $\gamma = 0.001$.

5.1.1.4 Instance Center Decoder

The last decoder head is the object center confidence head, which is analogous to the other two in terms of architecture design. This branch predicts, for each grid element, the confidence of it being the center of an object. In this decoder, the output volume is of depth 1, followed by a sigmoid activation of shape $\hat{y} = \sigma(\text{logit}) = 1/(1 + e^{-\text{logit}})$. This branch is optimized by a weighted cross-entropy loss between the output and the object center targets. However, because the number of grid elements is orders of magnitude larger than the average amount of objects in each image, the easy background elements overwhelm the loss. Therefore, we follow Lin *et al.* [92], and add an extra focal loss term modulated by γ, in Equation (5.7). This γ modulation makes the loss of easy background examples lower to circumvent the overwhelming of the loss.

$$L_{\text{centers}} = \begin{cases} -\alpha\ (1-\hat{y})^{\gamma}\ \log(\hat{y}) & \text{if } y = 1 \\ -(1-\alpha)\ \hat{y}^{\gamma}\ \log(1-\hat{y}) & \text{if } y = 0 \end{cases} \tag{5.7}$$

In our experience, the object loss does not converge unless this term is added. Another important parameter for the confidence head is the initialization of the bias of the last layer before the sigmoid. Usually, models for binary classification are initialized to output positive or negative class with equal probability, and therefore, in the presence of imbalance as extreme as our task, the loss is dominated by the easy negatives causing instabilities. Following [92], we initialize the last bias of this decoder to $b = -\log((1-\pi)/\pi)$, where π is a prior calculated from the class imbalance. For all experiments, we use $\pi = 0.1$, $\alpha = 0.01$ and $\gamma = 2$, which are obtained as the parameter which gives the lowest cross-validation error.

5.1.2 Postprocessing

Once the CNN has predicted all three heads, we perform a fast post-processing step to obtain the final output. First, we mask the embeddings and object centers with the semantic segmentation grid from the first head, to separate the embeddings and centers of each class from each other and the background, see Figure 5.2. Then we extract all grid elements that have center confidence over 0.7 and extract the center of mass of all connected components in this binary mask. This step is necessary because the center of an object is not a perfectly defined concept, and therefore the CNN usually outputs blobs instead of single elements for each instance. We also perform an elimination of "duplicated" cluster centers, which are the centers that have embedding distances lower than δ_a from Equation (5.3). These steps are analogous to the non-maximum suppression step in object detectors, where several anchors detect an object, and only one is kept by setting a threshold in the intersection over union.

Once we extract all the centers of objects, we mask their 32-dimensional embeddings as well as all the embeddings of the grid elements belonging to a certain class, and we calculate a similarity matrix between all elements in the class and all the centers. The next step calculates the maximum similarity object center for each element and filters the ones that have a distance greater than δ_a, to eliminate elements that were incorrectly assigned by the semantic head. This procedure assigns a unique id to each instance in a class and groups all their elements and is repeated for all classes. The final step in the post-processing is to upsample each grid element for both the clustered instances and the semantic mask into the original size output space, which is done using the one-to-many mapping exploiting the neighborhood color information from the superpixels, see Figure 5.2. We explain how we obtain this mapping in the following section.

5.1.3 Superpixel Extraction for Locally Consistent Upsampling

Our approach upsamples the low-resolution output grids of the CNN using an over-segmentation grid of the input in superpixels, improving over simple bilinear upsampling without sacrificing runtime. These superpixels need to be small enough to capture the "connectivity" of real-world objects, avoiding under-segmentation, but as big as possible to maximize the reduction of the size of the CNN output. This makes both, the inference and the clustering of the instances, faster. Thus, this size is a compromise which depends on the sizes of objects in the data, and is selected by evaluating the under-segmentation error in the training set of our data, the Cityscapes dataset [30].

For an image size $H \times W$ and a superpixel size k, the SLIC algorithm starts with a grid of size $H/k \times W/k$ of evenly distributed cluster centers c. The approach then iteratively (i) obtains the nearest neighbor cluster center using a distance

$$\delta_s = \|\mathcal{I}_{\text{Lab}}(i) - \mathcal{I}_{\text{Lab}}(c)\| + \alpha \|\mathcal{I}_{\text{xy}}(i) - \mathcal{I}_{\text{xy}}(c)\| \tag{5.8}$$

for each pixel i, only looking in the clusters present in a $2K \times 2K$ neigborhood; and (ii) updates the cluster centers to become the mean of the newly associated cluster pixels. This is done until convergence, or until a limit number of iterations is met. Thanks to its definition as a selective nearest neighbor search followed by cluster center update, SLIC is highly parallelizable. The gSLICr implementation used in our approach performs the nearest neighbor search in CUDA using one thread per pixel, as well as the cluster update using a kernel to do the accumulation of the energy values, and a separate one for the reduction which returns the updated clusters. This allows for a segmentation that runs around 83 times faster than its CPU counterpart. Due to the rigid nature of the output of all CNNs, if we want to use the extracted superpixels to improve the boundary consistency of the output masks using a one-to-many mapping, we need all of the cluster centers in the over-segmentation to remain in a grid-like structure. Therefore, we perform one iteration of the algorithm to get the boundary information of each locally consistent area, but without updating the cluster centers. This yields a grid in a structure that has a close resemblance with the regular grid with which the algorithm was initialized.

5.2 Experimental Evaluation

The experiments are designed to show the efficiency and performance of our joint semantic and instance segmentation approach, and to support the claims that our method is capable of performing both tasks in parallel, accurately, and faster

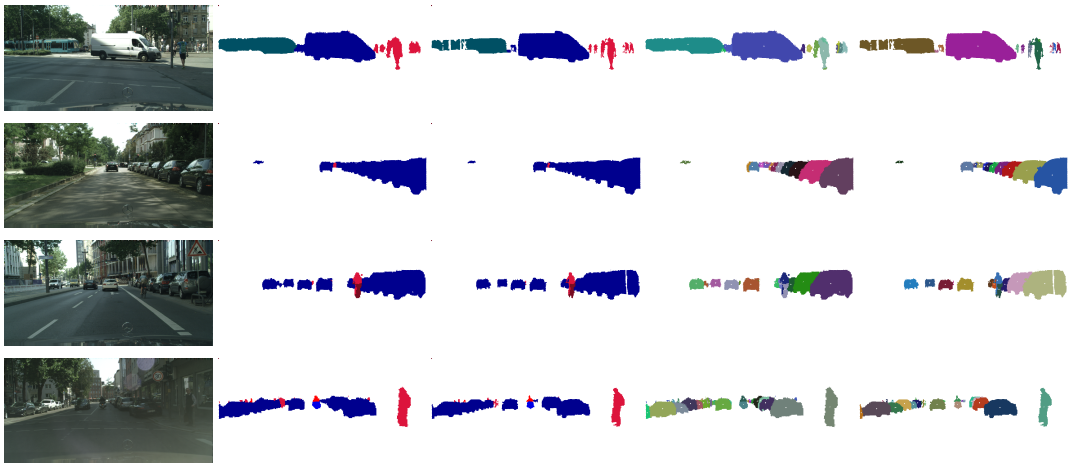

Figure 5.4: Examples of results on the validation set for our MobilenetsV2 backend using a decoder output grid downsampled 16 times (decoder OS 4).

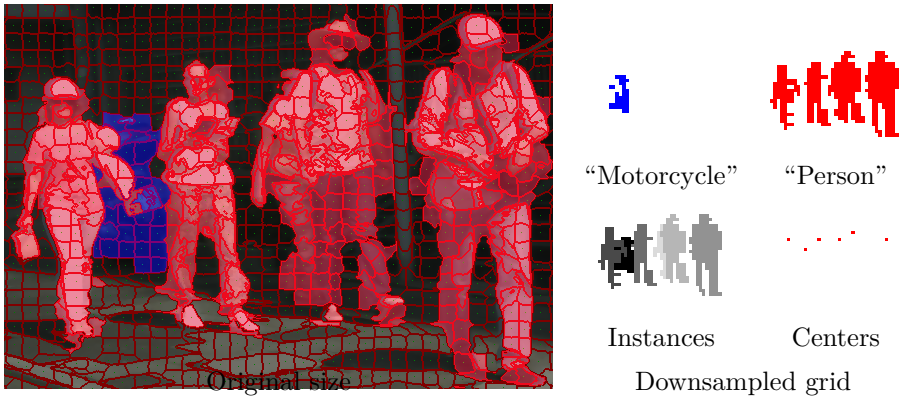

Figure 5.5: Procedure to get from original sized labels to downsampled versions using the label at the center of mass of each superpixel.

than previous approaches. We implemented the whole approach presented in this work relying on Pytorch.

5.2.1 Training Data

For our experiments, we use the Cityscapes dataset [30], which contains 5 000 annotated images with fine annotations of 30 semantic classes, 8 of which contain instance labels, plus 20 000 images containing coarse annotations. For our experiments, we use the 8 classes containing fine pixel-wise instance information: "person", "rider", "car", "truck", "bus", "train", "motorcycle", and "bicycle", and we treat the remaining pixels as background, both for the semantic as well as the instance task, see Figure 5.4 for examples of the images in the dataset. To train our architecture, we choose a downsampling rate based on two crite-

Table 5.1: Performance on validation set by encoder and output stride

Encoder	Decoder OS	Instance mAP@50%	Instance mAP	Semantic IoU	Runtime (avg.) Spix	CNN	Cluster	FPS
Baseline using Darknet 53	1	46.9%	24.3%	65.4%	-	138 ms	240 ms	2
Ours using Darknet 53	1	46.7%	22.3%	65.4%	-	141 ms	36 ms	5.5
	4	46.5%	21.9%	63.7%	4 ms	73 ms	9 ms	12
Ours using Mobilenets V2	1	46.2%	21.9%	62.3%	-	48 ms	36 ms	11
	4	45.2%	21.1%	60.4%	4 ms	19 ms	9 ms	31

ria: maximizing the superpixel size to decrease the output stride of the decoders, making the clustering of the instance elements faster, and at the same time, minimizing the under-segmentation error. For all experiments, we use an image size of 1024×512 and the best performing superpixel size in our experiments was 4, which allows us to reduce the number of pixels in the post-processing by 16, as well as reduce the inference time of the CNN. We train our networks downsampling the targets using the superpixels extracted from the inputs. This is if an image and its corresponding semantic and instance label are of size $H \times W$, and the superpixels are of size k, the output grids and the used targets for each loss are of size $H/k \times W/k$ and each target grid element is the label at the center of mass of its corresponding superpixel, see Figure 5.5. We train the whole architecture end-to-end starting with the encoder pre-trained on ImageNet, as stated in Section 5.1.1.

5.2.2 Performance

This experiment is designed to show that our algorithm can efficiently segment and classify individual objects from camera images, without sacrificing accuracy.

Table 5.1 shows the results of training in the 2,975 training set images with fine, pixel-wise annotations, and validating the results in 500 held out images from different cities. The first row shows the results of training an architecture with a strong, state of the art backbone (DN53) and inferring only the semantic and embedding branch, without the embedding centers or superpixel upsampling, and applying an all-to-all clustering of the instance pixels afterward. The subsequent rows show our approach inferring the cluster centers with different backbones, and decoder output strides. The results show two interesting effects. First, even though the baseline with no superpixel upsampling or center inference performs slightly better than our best performing architecture (row 0 vs. 1), this costs almost 3 times more to run, due to the expensive post-processing in the absence of the inferred cluster centers. Second, when using decoder OS 4, which means upsampling with superpixels of size $k = 4$, the models perform similarly to their non-upsampled counterparts but run 2 times faster in the case of the Darknet

Table 5.2: Comparison with other state-of-the-art real-time semantic segmentation methods.

IoU	Segnet*[7]	ERFNet*[151]	ENet*[126]	Ours MNv2	Ours DN53
Car	89.2%	93.4%	91.0%	88.6%	94.1%
Bus	43.1%	60.8%	49.3%	77.3%	73.2%
Truck	38.1%	52.2%	39.3%	57.4%	62.1%
Motorc.	35.7%	49.8%	41.6%	38.9%	45.8%
Train	44.1%	53.7%	50.5%	61.9%	62.4%
Bicyc.	51.8%	64.2%	59.8%	48.6%	53.9%
Person	62.7%	78.5%	71.3%	60.2%	61.3%
Rider	42.8%	59.7%	49.6%	50.6%	57.3%
Mean	50.9%	64.0%	56.5%	60.4%	63.7%
FPS	16	50	76	31	12

Methods with * perform the semantic task exclusively, not predicting instances.

Table 5.3: Comparison with other state-of-the-art instance segmentation methods.

Model	AP	AP@50%	FPS
DeepWatershed[8]	19.4%	35.3%	-
FSUfAD[121]	21.0%	38.6%	21
Mask-RCNN [59]	26.2%	49.9%	2
PANet[94]	31.8%	57.1%	2
Mask-RCNN* [59]	32.0%	58.1%	2
PANet*[94]	36.4%	63.1%	2
Ours MNv2	21.1%	45.2%	31
Ours DN53	21.9%	46.5%	12

Methods with * were pretrained with COCO [91] dataset.

based model, and almost 3 times faster for the Mobilenets based one.

Table 5.2 and Table 5.3 show a comparison to other state-of-the-art methods for semantic segmentation and instance segmentation, respectively. Table 5.2 shows that our algorithm performs the semantic segmentation task on par with other state-of-the-art, real-time methods, but with the burden of having to infer the instances as well. Table 5.3 shows that our approach performs better than other state of the art methods, but slightly under-performs the best benchmark submissions such as MaskRCNN [59] and PANet [94], which are roughly 15 times slower to run.

5.3 Related Work

Several methods for instance segmentation have been proposed over the last years, most of them based on some CNN backbone for feature extraction but using different types of decoders. Such approaches can be grouped according to how they exploit different decoder architectures.

One group of approaches works with *region proposals* coming from an object detection pipeline and a posterior mask segmentation between foreground and background. This has the advantage that these approaches can deal with an arbitrary object number and usually provide accurate results. State-of-the-art examples are Mask-RCNN [59] or PANet [94]. Such type of pipeline is particularly useful in application-specific use cases where obtaining region proposals is easy, for example in the approach presented in Chapter 3. However, there are two main disadvantages of these types of proposal-based, two-stage pipelines. First, they are usually slow, and second, the relationship between the object mask and its bounding box may not be straight-forward. This may render the computation of the object mask hard in some cases. Furthermore, most of these approaches infer the segmentation of the foreground class in each bounding box proposal in a reduced resolution, relying on bilinear upsampling to map it back to the original image size. This sacrifices the accuracy of the object boundaries.

The second group of approaches relies on *recurrent neural networks* and *attention mechanisms* [148, 152]. They take a single image as an input and output instance masks, one by one in a counting-like manner. This is sometimes reported as biologically inspired, but these approaches do not offer the best accuracy and runtime.

The last group of approaches to instance segmentation is based on *metric learning* strategies [8, 16, 41, 121, 128]. They rely on predicting a discriminative high-dimensional embedding for each pixel, followed by clustering of pixel embeddings into individual instances. Some methods in this context rely on a previously executed, semantic segmentation of the input into the individual semantic classes, and subsequently predict an embedding [16] or an energy function [8] that allows to separate the pixels inside each semantic class mask into individual instances. The required semantic segmentation usually comes from a separate, and computationally expensive CNN and, thus, except for a few exceptions, these approaches cannot run in real-time on a robot, and their accuracy has an upper bound that is given by the performance of the pre-segmentation CNN. Alternative approaches [41, 121], try to predict both, the class labels, as well as the instance embeddings at the same time. There is, however, no concept of "objectness" in these types of pipelines, which makes the posterior clustering of the embeddings difficult and computationally demanding.

Our approach overcomes this limitation by adding the inference of the object centers' confidence such as in one-shot object detection pipelines, which in turn allows us to speed up the post-processing by avoiding the expensive all-to-all similarity matrix calculation. Furthermore, we exploit fast GPU-based superpixel summarization, which exploits the local connectivity of pixels in objects, and allows us to further reduce the number of computations needed to calculate the

similarity matrix between the embeddings in each semantic class and the centers of clusters. This is key for achieving semantic analysis at the camera frame-rate.

5.4 Conclusion

In this chapter, we presented a novel approach for joint semantic and instance segmentation in an efficient manner. Our main contribution is a CNN architecture based on one-shot object detection and metric learning that can perform the semantic and instance segmentation tasks simultaneously in real-time. We achieve this by performing a fast GPU-based superpixel over-segmentation, which allows us to exploit local connectivity of neighboring pixels and reduce the complexity of our algorithm, while still obtaining high-resolution masks. Our one-shot object detection based pipeline jointly predicts the confidence of each superpixel being the center of an object, its semantic class, and a high-dimensional embedding which allows us to assign each individual occupied superpixel to an instance center. This allows us to achieve three important algorithmic qualities. Firstly, given the over-segmentation grid and the regression of the instance centers as object confidences, we avoid the selection of the number of cluster centers for the k-means clustering of the embeddings, and instead we can just count the regressed instance centers over some desired confidence level. Secondly, by exploiting the neighborhood information we both reduce the number of pixels to cluster, allowing us to use bigger images, and avoid stranded pixels at test time. Finally, the one-shot architecture combined with the neighborhood summarization makes the extraction of the instances faster when compared to two-stage as well as other metric learning-based methods.

Part II

Scene Understanding using LiDAR Sensors

Chapter 6

Large-Scale Supervised Learning of LiDAR data

Semantic scene understanding is an integral part of any self-driving car's software stack. Particularly, fine-grained understanding provided by semantic, as well as panoptic segmentation is necessary to distinguish drivable and non-drivable surfaces and to reason about functional properties, like parking areas and sidewalks, as showcased in Chapter 5. Currently, such understanding, represented in so-called high definition maps, is mainly generated in advance using surveying vehicles. However, self-driving cars should also be able to drive in unmapped areas and adapt their behavior if there are changes in the environment. Most self-driving cars currently use multiple different sensors to perceive the environment. Complementary sensor modalities enable to cope with deficits or failures of particular sensors. Besides cameras, light detection and ranging (LiDAR) sensors are often used as they provide precise distance measurements that are not affected by lighting.

Publicly available datasets and benchmarks are crucial for empirical evaluation of research. They mainly fulfill three purposes: (i) they provide a basis to measure progress since they allow to provide results that are reproducible and comparable, (ii) they uncover shortcomings of the current state of the art and therefore pave the way for novel approaches and research directions, and (iii) they make it possible to develop approaches without the need to first painstakingly collect and label data. While multiple large datasets for *image-based* semantic and instance segmentation exist, such as the ones used in Part I [30, 119, 22], publicly available datasets with point-wise annotation of three-dimensional point clouds at the time this work was done were comparably small, as shown in Table 6.1. It is important to notice that since the release of our benchmark, multiple other LiDAR datasets have been released which also exploit the sequentiality of the data for autonomous driving [18, 74, 167], highlighting the topic's relevance.

	#scans[1]	#points[2][M]	#classes[3]	annotation	sequential
SemanticKITTI	**23k/20k**	**4 549**	**25** (28)	point-wise	✓
Oakland3d [118]	17	1.6	5 (44)	point-wise	✗
Freiburg [165, 13]	77	1.1	4 (11)	point-wise	✗
Wachtberg [13]	5	0.4	5 (5)	point-wise	✗
Semantic3d [55]	15/15	4 009	8 (8)	point-wise	✗
Paris-Lille-3D [156]	3	143	9 (50)	point-wise	✗
Zhang et al. [192]	140/112	32	10 (10)	point-wise	✗
A2D2*[48]	38k	5 772**	38	point-wise	✗
PandaSet*[160]	16k	1 920	37	point-wise	✓
KITTI [46]	7k/7k	1 799	3	bounding box	✗
Waymo* [167]	230k	40 710	4	bounding box	✓
NuScenes* [18]	40k	1 360	23	bounding box	✓
Lyft L5* [74]	46k	9 960	9	bounding box	✓

Table 6.1: Overview of other point cloud datasets with semantic annotations. Ours is by far the largest dataset with sequential information. [1]Number of scans for train and test set, [2]Number of points is given in millions, [3]Number of classes used for evaluation and number of classes annotated in brackets. Datasets marked with * post-date our work. Furthermore, in datasets marked with **, only the points which overlap with camera FOV are semantically segmented.

To close the gap between the data starvation for machine learning algorithms using LiDAR and researchers in the field, we took the effort to produce this type of data and proposed SemanticKITTI. SemanticKITTI is a large sequential dataset showing unprecedented detail in point-wise annotation with 28 classes, which is suited for various tasks. Furthermore, for 8 of the potentially-moving classes, such as car, person, etc., we also provide different labels for moving and non-moving instances, as well as temporally consistent instance IDs. This allows to perform, besides semantic segmentation, also temporally consistent panoptic segmentation, as ventured by Hurtado *et al.* [69].

In this chapter, we will mainly focus on *laser-based* semantic as well as panoptic segmentation. The dataset is distinct from other laser datasets as we provide accurate scan-wise annotations of sequences. Overall, we annotated all 22 sequences of the odometry benchmark of the KITTI Vision Benchmark [46] consisting of over 43 000 scans. Moreover, we labeled the complete horizontal 360° field-of-view of the rotating laser sensor. Figure 6.1 shows example scenes with our semantic segmentation labels, and Figure 6.2 shows examples of the consistent instance annotation.

This large dataset has stimulated the development of novel algorithms, as it makes it possible to investigate new research directions, and puts evaluation and comparison of these novel algorithms on a more solid ground [9].

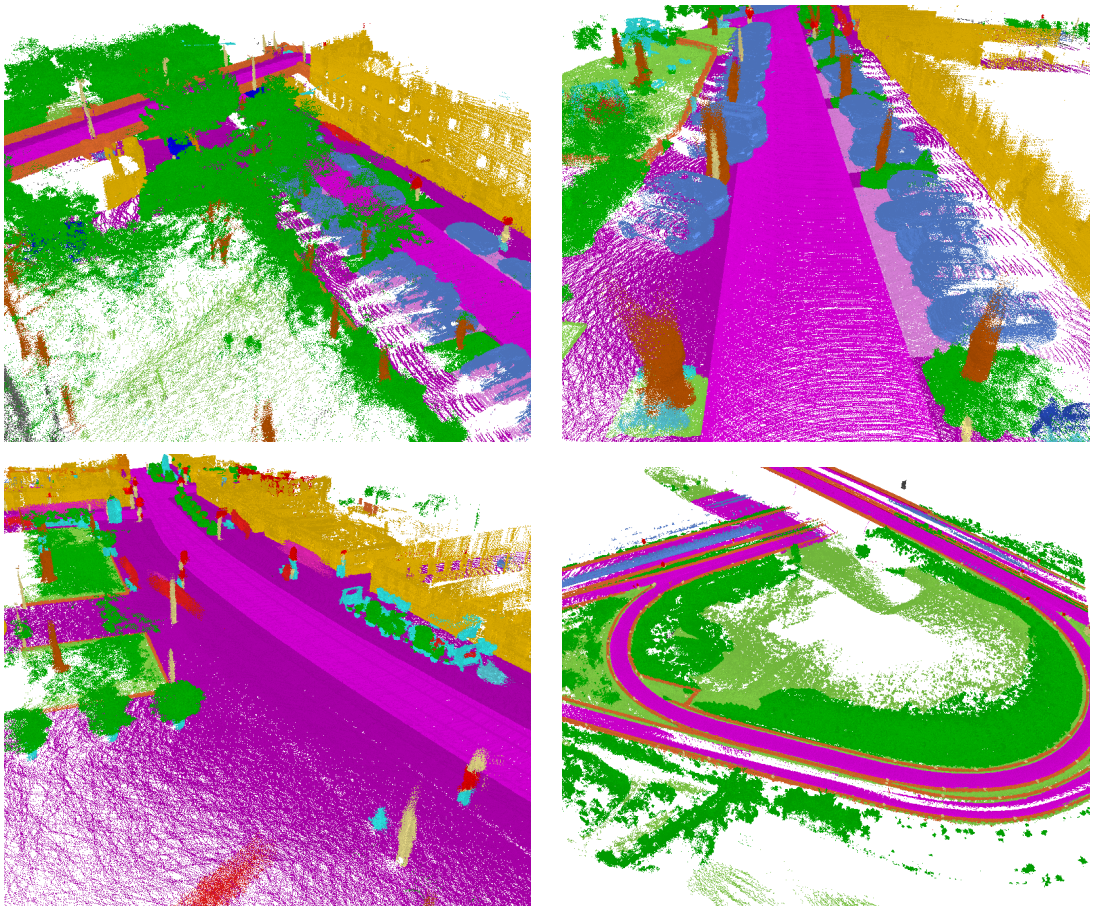

Figure 6.1: Our dataset provides dense annotations for each scan of all sequences from the KITTI Odometry Benchmark [46]. Here, we show multiple scans aggregated using pose information estimated by a SLAM approach.

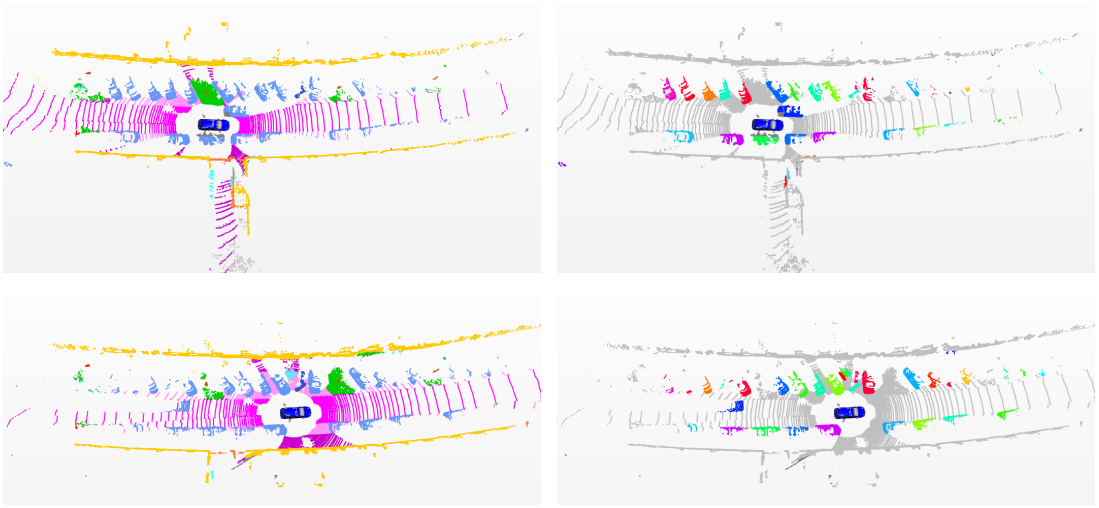

Figure 6.2: Our dataset also provides the instance annotations over a sequence of scans: on the left is the semantic annotation and on the right is the instance annotation shown. Note, same colors at different timestamps correspond to the same instance id. Along with the semantic labels, this allows for temporally-consistent panoptic segmentation. Best viewed in color.

6.1 The SemanticKITTI Dataset

Our dataset is based on the odometry dataset of the KITTI Vision Benchmark [46] showing inner-city traffic, residential areas, but also highway scenes and countryside roads around Karlsruhe, Germany. The original odometry dataset consists of 22 sequences, splitting sequences 00 to 10 as the training set, and 11 to 21 as the test set. For consistency with the original benchmark, we adopt the same division for our training and test set. Moreover, we do not interfere with the original odometry benchmark by providing labels only for the training data. Overall, we provide 23 201 full 3D scans for training and 20 351 for testing, which, even after the release of multiple other datasets, is still the largest dataset publicly available for LiDAR-only semantic and panoptic segmentation. Furthermore, it is the only one including entire complex sequences, including loop closures, and multiple passes of an area, which enables the study of how semantics affect the Simultaneous Localization and Mapping (SLAM) process [27].

We decided to use the KITTI dataset as a basis for our labeling effort since it allowed us to exploit one of the largest available collections of raw point cloud data captured in a car. We furthermore expect that there are also potential synergies between our annotations and the existing benchmarks and this will enable the investigation and evaluation of additional research directions, such as the aforementioned usage of semantics for laser-based odometry estimation [27].

Compared to other datasets, as shown in Table 6.1, we provide labels for sequential point clouds generated with a commonly used automotive LiDAR, i.e., the Velodyne HDL-64E. Other publicly available datasets, like *Paris-Lille-3D* [156] or *Wachtberg* [13], also use such sensors, but only provide the aggregated point cloud of the whole acquired sequence or some individual scans of the whole sequence, respectively. Since we provide the individual scans of the whole sequence, one can also investigate how aggregating multiple consecutive scans influences the performance of the semantic segmentation and use the information to recognize moving objects.

We annotated 28 classes, where we ensured a large overlap of classes with the *Mapillary Vistas* dataset [119] and *Cityscapes* dataset [30] and made modifications where necessary to account for the sparsity and vertical field-of-view. More specifically, we do not distinguish between persons riding a vehicle and the vehicle but label the vehicle and the person as either *bicyclist* or *motorcyclist*.

We furthermore distinguished between moving and non-moving vehicles and humans, i.e., vehicles or humans gets the corresponding moving class if they moved in some scan while observing them, as shown in the lower part of Figure 6.5. For all of these, we also assigned a temporally consistent instance id. All annotated classes and their statistics are listed in Figure 6.3, and the statistics

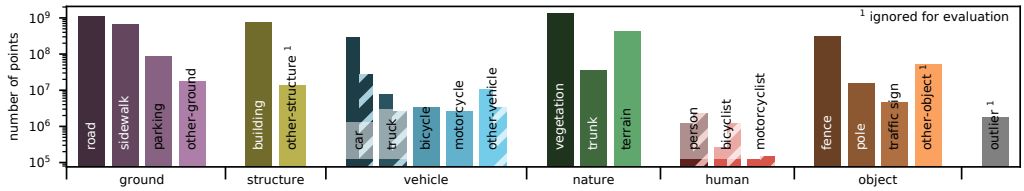

Figure 6.3: Semantic Segmentation label distribution. The number of labeled points per class and the root categories for the classes are shown. For movable classes, we also show the number of points on non-moving (solid bars) and moving objects (hatched bars).

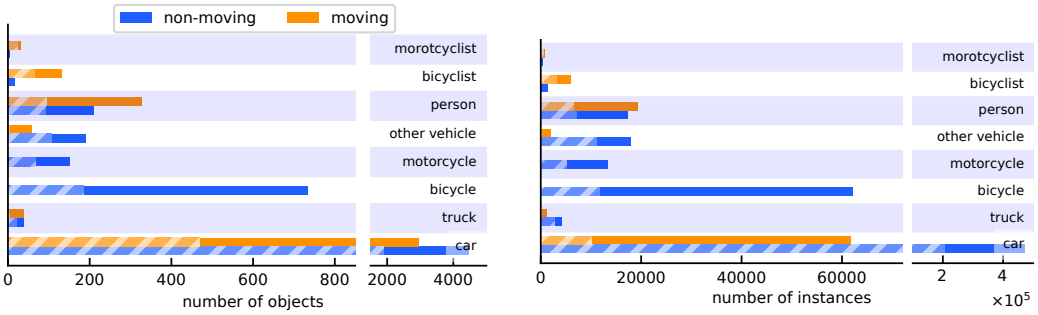

Figure 6.4: Instance label distribution. Left: number of (sequence-wise) objects. Right: number of (scan-wise) instances. The hashed bars correspond to the training data. The large number of scan-wise annotations in relation to the number of objects indicates that many objects are seen over an extended period of time.

of the instance labels are present in Figure 6.4. In summary, we have 28 classes, where 6 classes are assigned the attribute moving or non-moving as well as an instance id, and one *outlier* class is included for erroneous laser measurements caused by reflections or other effects.

The dataset is publicly available through a benchmarking website and we provide only the training set with ground truth labels and perform the test set evaluation online. We furthermore will also limit the number of possible test set evaluations to prevent overfitting to the test set [175]. In the said website, two benchmarks are available: a semantic segmentation benchmark and a panoptic segmentation benchmark. In summary, our main contributions are:

- We present a point-wise annotated dataset of point cloud sequences with an unprecedented number of classes and unseen level-of-detail for each scan.
- We provide temporally-consistent instance annotations for all traffic participants including vehicles, pedestrians, bicyclists, and motorcyclists for the KITTI Odometry Benchmark.
- We provide two benchmarks for semantic and panoptic segmentation which allow researchers in the field to test their approaches in a hidden test set, which is also consistent with the split of a well-established odometry benchmark, allowing for synergies between these [27, 46]

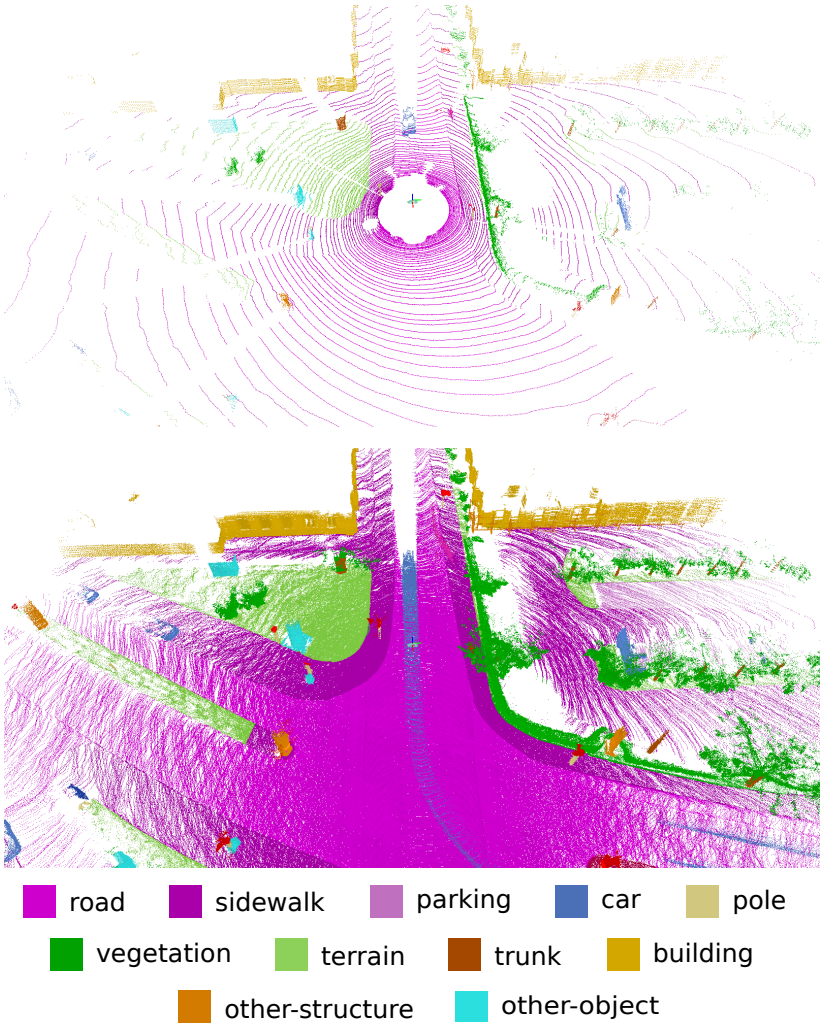

Figure 6.5: Single scan (top) and multiple superimposed scans with labels (bottom). Also shown is a moving car in the center of the image resulting in a trace of points.

6.2 Semantic Segmentation

6.2.1 Labeling Process

To make the labeling of point cloud sequences practical, we superimpose multiple scans above each other, which conversely allows us to label multiple scans consistently. To this end, we first register and loop close the sequences using an off-the-shelf laser-based SLAM system [12]. This step is needed as the provided information of the inertial navigation system (INS) often results in map inconsistencies, i.e., streets that are revisited after some time have different height. For three sequences, we had to manually add loop closure constraints to get correctly loop closed trajectories, since this is essential to get consistent point clouds for annotation. The loop closed poses allow us to load all overlapping point clouds for specific locations and visualize them together, as depicted in Figure 6.5.

We subdivide the sequence of point clouds into tiles of 100 m by 100 m. For each tile, we only load scans overlapping with the tile. This enables us to label all scans consistently even when we encounter temporally distant loop closures. To ensure consistency for scans overlapping with more than one tile, we show all points inside each tile and a small boundary overlapping with neighboring tiles. Thus, it is possible to continue labels from a neighboring tile.

Following best practices, we compiled a labeling instruction and provided instructional videos on how to label certain objects, such as cars and bicycles standing near a wall. Compared to image-based annotation, the annotation process with point clouds is more complex, since the annotator often needs to change the viewpoint. An annotator needs on average 4.5 hours per tile, when labeling residential areas corresponding to the most complex encountered scenery, and needs on average 1.5 hours for labeling a highway tile.

We explicitly did not use bounding boxes or other available annotations for the KITTI dataset, since we want to ensure that the labeling is consistent and the point-wise labels should only contain the object itself.

We provided regular feedback to the annotators to improve the quality and accuracy of labels. Nevertheless, a single annotator also verified the labels in a second pass, i.e., corrected inconsistencies and added missing labels. In summary, the whole dataset comprises 518 tiles and over 1 400 hours of labeling effort have been invested with additional $10-60$ minutes verification and correction per tile, resulting in a total of over 1 700 hours.

6.2.2 Dataset Statistics

Figure 6.3 shows the distribution of the different classes, where we also included the root categories as labels on the x-axis. The ground classes, *road*, *sidewalk*, *building*, *vegetation*, and *terrain* are the most frequent classes. The class *motorcyclist* only occurs rarely, but still more than 100 000 points are annotated.

The unbalanced count of classes is common for datasets captured in natural environments and some classes will be always under-represented since they do not occur that often. Thus, an unbalanced class distribution is part of the problem that an approach has to master. Overall, the distribution and relative differences between the classes is quite similar in other datasets, e.g., *Cityscapes* [30].

6.2.3 Evaluation Metrics

In semantic segmentation of point clouds, we want to infer the label of each three-dimensional point. Therefore, the input to all evaluated methods is a list of coordinates of the three-dimensional points along with their remission, i.e., the strength of the reflected laser beam which depends on the properties of the

surface that was hit. Each method should then output a label for each point of a scan, i.e., one full turn of the rotating LiDAR sensor.

To assess the labeling performance, we rely on the commonly applied mean Jaccard Index or mean intersection-over-union (mIoU) metric [40] over all classes, given by

$$\text{mIoU} = \frac{1}{C} \sum_{c=1}^{C} \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c + \text{FN}_c}, \tag{6.1}$$

where TP_c, FP_c, and FN_c correspond to the number of true positive, false positive, and false negative predictions for class c, and C is the number of classes.

As the classes *other-structure* and *other-object* have either only a few points and are otherwise too diverse with a high intra-class variation, we decided to not include these classes in the evaluation. Thus, we use 25 instead of 28 classes, ignoring *outlier*, *other-structure*, and *other-object* during training and inference.

Furthermore, we cannot expect to distinguish moving from non-moving objects with a single scan, since this Velodyne LiDAR cannot measure velocities like radars exploiting the Doppler effect. We, therefore, combine the moving classes with the corresponding non-moving class resulting in a total number of 19 classes for training and evaluation. This is the metric and list of classes used in the single scan semantic segmentation task of the semantic segmentation benchmark. Another benchmark task for multi-scan semantic segmentation is provided which differentiates between moving and non-moving objects.

6.3 Panoptic Segmentation

6.3.1 Labeling Process

For annotation of the instances that allow the panoptic segmentation task, we employ a semi-automatic process using different strategies to generate temporally consistent instance annotation. Our goal is to label the same instance through the whole sequence with the same instance ID – even for instances that move. For static objects, the data association can be simply performed by considering the location of the segment after performing a pose correction using a SLAM system [12]. For moving objects, we have to account for the motion of the object as well as the motion of the sensor at the same time.

Overall, our dataset provides 28 classes (including 6 classes to distinguish moving from non-moving classes) from which we select the traffic participants as *thing* classes for the panoptic segmentation, i.e., car, truck, other-vehicle, motorcycle, bicycle, person, bicyclist, and motorcyclist. The remaining classes are *stuff*

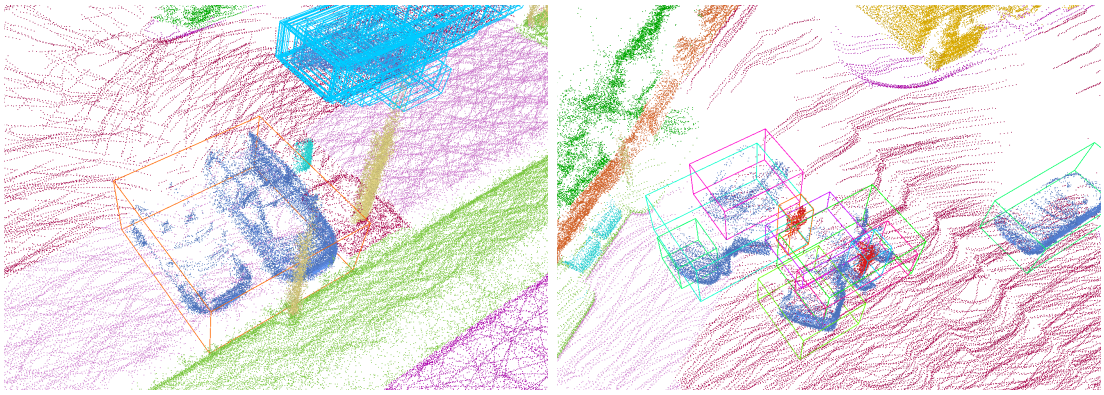

Figure 6.6: Example of under- (left) and over-segmentation (right) generated by our semi-automated clustering approach.

classes for the panoptic segmentation, i.e., road, sidewalk, parking, other-ground, building, vegetation, trunk, terrain, fence, pole, and traffic-sign.

For static thing classes, we first cluster all points for each individual class using a fast grid-based segmentation approach [14] to handle a large number of points efficiently. We then split the aggregated point cloud into tiles of size 100 m by 100 m using the pose information by our SLAM system [12]. For each tile, we use a two-dimensional grid with cell size 0.1 m by 0.1 m, which allows us often to separate even close parking cars. Next, all points are inserted into the corresponding grid cells using their x and y-coordinates. Finally, only grid cells with points exceeding a height threshold of $\Delta > 0.5$ m are combined using a flood fill algorithm to combine neighboring grid cells into segments.

For moving thing classes, we generate clusters for each scan individually using a distance-based clustering as this provided more reliable results and could be also used to associate instances between consecutive scans. First, we search for each point in a radius of 0.5 m for the nearest neighbor, and cluster points together that share neighbors. To find associations with the previous 4 scans, we use a slightly larger radius of 1.0 m to find neighbors between two different timestamps. If we find enough neighbors with the previous segments at different timestamps, we associate them together and assign the same instance ID.

The described clustering leads inevitably to over- and under-segmentation, shown in Figure 6.6, but also to wrong or missing associations between consecutive timestamps. We correct these issues manually using an own point labeling tool, which provides tools to create, join, and split instances. Overall, the manual correction for all 22 sequences took roughly 70 h of additional labor.

6.3.2 Dataset Statistics

Figure 6.4 provides an overview of the number of instances and the actual number of bounding boxes per class. We show in the upper part the sequence-wise counts of instance annotations, i.e., we count each object only once even if it is seen multiple times by the sensor. The lower part of the figure shows the accumulated scan-wise counts of instances, where we count the instances without considering the temporally consistent instance ID.

The bulk of the instances correspond to cars, which are naturally occurring in city-like environments and also correspond to the normal statistics in autonomous driving scenarios. Usually, an autonomous car will also encounter some classes far fewer than other classes or situations. They are usually denoted as the "long tail" problem, referring to the underrepresented entities in a given distribution. This adds complexity to the task, since panoptic segmentation approaches, which are designed to tackle this scenario must be able to deal with such skewed class distributions.

6.3.3 Evaluation Metrics

In panoptic segmentation, each point $\mathbf{p}_i$ not only caries a class label $y_i \in \mathcal{Y}$, where $|\mathcal{Y}|$ is the number of classes, but also can have an instance ID n_i, where $n_i = 0$ denotes no specific instance.

To measure the quality of this joint assignment, we briefly recapitulate the recently proposed panoptic quality (PQ) metric [76]. Let $\mathcal{S}, \hat{\mathcal{S}}$ denote segments, i.e., sets of points in our specific case, sharing an class and instance ID. Here, we assume that the *stuff* classes, e.g., vegetation, simply get instance ID $n_i = 0$ corresponding to no specific instance assigned.

Furthermore, let $\mathrm{IoU}(\mathcal{S}, \hat{\mathcal{S}}) = (\mathcal{S} \cap \hat{\mathcal{S}})(\mathcal{S} \cup \hat{\mathcal{S}})^{-1}$ denote the intersection-over-union of these two sets. Let the set of true positive matches TP_c be the pairs of predicted segments $\hat{\mathcal{S}}$ that overlap at least with 0.5 IoU with a ground truth segment $\mathcal{S}$, $\mathrm{TP}_c = \{(\mathcal{S}, \hat{\mathcal{S}}) \mid \mathrm{IoU}(\mathcal{S}, \hat{\mathcal{S}}) > 0.5\}$. Likewise, let FP_c the set of unmatched predicted segments $\hat{\mathcal{S}}$ and FN_c the set of unmatched ground truth segments $\mathcal{S}$.

With the above definitions, the class-wise PQ_c is given by

$$\mathrm{PQ}_c = \frac{\sum_{(\mathcal{S},\hat{\mathcal{S}})\in \mathrm{TP}_c} \mathrm{IoU}(\mathcal{S}, \hat{\mathcal{S}})}{|\mathrm{TP}_c| + \frac{1}{2}|\mathrm{FP}_c| + \frac{1}{2}|\mathrm{FN}_c|}. \tag{6.2}$$

The panoptic quality metric is computed for each class independently and averaged over all classes, which makes the metric insensitive to class imbalance [76],

i.e.,

$$\mathrm{PQ} = \frac{1}{|\mathcal{Y}|} \sum_{c \in \mathcal{Y}} \mathrm{PQ}_c. \tag{6.3}$$

For images, Porzi *et al.* [134] proposed to alter the metric to account for *stuff* classes having only a single segment since no pixels (or, in our case, points) have an instance ID. In such a case, the IoU-based criterion could often lead to an unmatched prediction. To account for *stuff* classes, Porzi *et al.* use

$$\mathrm{PQ}_c^{\dagger} = \begin{cases} \mathrm{IoU}(\mathcal{S}, \hat{\mathcal{S}}) & \text{, if c is a } \textit{stuff} \text{ class} \\ \mathrm{PQ}_c & \text{, otherwise.} \end{cases} \tag{6.4}$$

Consequently, we denote by $\mathrm{PQ}^{\dagger}$ the average over the class-wise modified $\mathrm{PQ}_c^{\dagger}$ as defined in Equation (6.3).

Furthermore, the quality of the semantic segmentation is also measured using the mean intersection-over-union (mIoU), which also enables the comparison with other approaches in the semantic segmentation benchmark. This metric is defined as follows:

$$\mathrm{mIoU} = \frac{1}{|\mathcal{Y}|} \sum_{c \in \mathcal{Y}} \frac{|\{i \mid y_i = c\} \cap \{j \mid \hat{y}_j = c\}|}{|\{i \mid y_i = c\} \cup \{j \mid \hat{y}_j = c\}|}, \tag{6.5}$$

where y_i corresponds to the ground truth label of point $\mathbf{p}_i$ and $\hat{y}_i$ to the prediction. This is the same metric described in Section 6.2.3 and used in its benchmark, and is also relevant for this task. This is especially true for the *stuff* classes.

We make use of all the metrics mentioned in the panoptic segmentation benchmark and provide a hidden test set evaluation with a limited number of submissions. Both the semantic as well as the panoptic segmentation benchmark can be found at `https://www.semantic-kitti.org`.

6.4 Related Work

Relevance of Open Datasets and Benchmarks

The progress of computer vision has always been driven by benchmarks and datasets [175], but the availability of especially large-scale datasets, such as *ImageNet* [34], was even a crucial prerequisite for the advent of deep learning. More task-specific datasets geared towards self-driving cars were also proposed. Notable is here the KITTI Vision Benchmark [46] since it showed that off-the-shelf solutions are not always suitable for autonomous driving.

The *Cityscapes* dataset [30] is the first dataset for self-driving car applications that provides a considerable amount of pixel-wise labeled images suitable for deep

learning. The *Mapillary Vistas* dataset [119] surpasses the amount and diversity of labeled data compared to *Cityscapes*.

Semantic Segmentation of Point Clouds

Also in point cloud-based interpretation, e.g., semantic segmentation, RGB-D based datasets enabled tremendous progress. *ShapeNet* [20] is especially noteworthy for point clouds showing a single object, but such data is not directly transferable to other domains. Specifically, LiDAR sensors usually do not cover objects as densely as an RGB-D sensor due to their lower angular resolution, in particular in the vertical direction.

For indoor environments, there are several datasets [163, 153, 66, 5, 32, 107, 88, 33] available, which are mainly recorded using RGB-D cameras or synthetically generated. However, such data shows very different characteristics compared to outdoor environments, which is also caused by the size of the environment, since point clouds captured indoors tend to be much denser due to the range at which objects are scanned. Furthermore, the sensors have different properties regarding sparsity and accuracy. While laser sensors are more precise than RGB-D sensors, they usually only capture a sparse point cloud compared to the latter.

For outdoor environments, datasets were recently proposed that are recorded with a terrestrial laser scanner (TLS), like the *Semantic3d* dataset [55], or using automotive LiDARs, like the *Paris-Lille-3D* dataset [156]. However, the *Paris-Lille-3D* provides only the aggregated scans with point-wise annotations for 50 classes from which 9 are selected for evaluation. Another recently used large dataset for autonomous driving [178], but with fewer classes, is not publicly available.

The *Virtual KITTI* dataset [43] provides synthetically generated sequential images with depth information and dense pixel-wise annotation. The depth information can also be used to generate point clouds. However, these point clouds do not show the same characteristics as a real rotating LiDAR, including defects like reflections and outliers.

Recently, almost all major self-driving car companies release datasets providing besides camera also LiDAR data, including Waymo [167], Lyft [74], Audi [48], Argo [21], and Aptiv [18]. While all datasets provide also the annotations for object instances by bounding boxes, only a few datasets provide point-wise semantic annotation [9, 48].
In contrast to these datasets, our dataset combines a large number of labeled points, a large variety of classes, and sequential scans generated by a commonly employed sensor used in autonomous driving, which is distinct from all publicly available datasets, also shown in Table 6.1.

Panoptic Segmentation of Point Clouds

Shortly after Kirillov *et al.* [76] proposed panoptic segmentation and a metric to measure the performance of approaches providing such labels, the established datasets for semantic segmentation of *image data*, i.e., Cityscapes [30], Microsoft's Common Objects in Context (COCO) [91], and Mapillary's Vistas [119] adopted the metric and added an evaluation for this task. The last version of the Joint COCO and Mapillary Recognition Challenge workshop at ICCV also featured a panoptic segmentation track.

Due to the availability of the data, we witnessed a wide adoption and interest for panoptic segmentation in the computer vision community [29, 76, 93, 132, 134, 186]. While there have also been approaches for RGB-D data [63, 132], it is only recently that approaches which operate on LiDAR data have surfaced [69], using SemanticKITTI, because before it was published there was no real annotated data available that provided point-wise semantic labels *and* instance information at the same time.

By providing now instance annotations together with an online evaluation on a hidden test set, we close the gap to the aforementioned established image-based dataset and provide means to evaluate panoptic segmentation using an automotive LiDAR. We hope that the availability of labeled LiDAR scans for panoptic segmentation opens the door for more research in the direction of LiDAR-based panoptic segmentation.

6.5 Conclusion

We presented SemanticKITTI, a large-scale dataset showing unprecedented scale in the point-wise annotation of point cloud sequences. We provide the labels that allow for training large deep learning models for the tasks of semantic and panoptic segmentation, as well as a hidden test set evaluation benchmark server to evaluate said approaches. In subsequent chapters, we will see application cases of both semantic and panoptic segmentation, both using this data.

Chapter 7

LiDAR Semantic Segmentation

As mentioned in all previous chapters, scene understanding is a key building block of autonomous robots working in dynamic environments. To obtain the required scene understanding, robots are often equipped with multiple sensors that allow them to leverage the strengths of each modality. Combining multiple complementary sensing modalities allows for covering the shortcomings of individual sensors such as cameras, laser scanners, or radars. This is particularly critical in the context of autonomous driving, where a failure of one modality can have significant monetary, or even lethal, consequences if not properly covered by another redundant modality.

An important task in semantic scene understanding is the task of semantic segmentation, which, as we know by now, assigns a class label to each data point in the input modality, i.e., to a pixel in case of a camera or to a 3D point obtained by a LiDAR. While the best approach to the task would use a combination of both, LiDAR and camera images, it is an interesting research enterprise to see how far LiDAR-only perception can be pushed, if the car was required to navigate under the complete absence of the other modality, e.g., if a sensor is down, or some weather or environmental condition blinds it operation completely. Therefore, in this chapter, we explicitly address semantic segmentation for rotating 3D LiDARs such as the commonly used Velodyne scanners.

One of the main problems limiting the development of LiDAR-only semantic segmentation models for autonomous driving was the fact that there was no large-scale openly available dataset for the task. As we discussed in Chapter 6, we eliminated this barrier by taking the initiative and providing the data ourselves, with a labeling effort of over a person-year. The majority of state-of-the-art methods currently available for semantic segmentation on point cloud data either do not have enough representational capacity to tackle the task, or are computationally too expensive to operate at frame-rate on a mobile GPU. Furthermore, since most were designed to receive uniformly sampled point clouds from explicit sur-

faces, they tend to perform poorly on non-uniform data such as a single LiDAR sweep. This makes them not suitable to aid the task of supporting autonomous vehicles, since offline processing is not possible for most tasks pertaining to autonomous vehicles, such as affordance analysis of the scene, localization, obstacle avoidance, etc. Presenting a new approach that addresses these shortcomings is the purpose of this chapter.

The main contribution of this work is a new method for accurate, fast, LiDAR-only semantic segmentation. We achieve this by operating on a spherical projection of the input point cloud, i.e., a 2D image representation, similar to a range image, and therefore exploit the way the points are detected by a rotating LiDAR sensor. Our method infers the full semantic segmentation for each pixel of the image using any CNN as a backbone. This yields an efficient approach but can lead to issues caused by discretization or blurry CNN outputs. We effectively resolve these issues via a reconstruction of the original point with semantics without discarding any points from the original point cloud, regardless of the used resolution of the image-based CNN. This post-processing step, which also runs online, operates on the image representation and is tailored towards efficiency. We can calculate nearest neighbors in constant time for each point and exploit GPU-based calculations. This allows us to infer full semantic segmentation of LiDAR point clouds accurately and faster than the frame rate of the sensor. Since the approach runs with any range image-based CNN backbone, we call it RangeNet++. This highlights that the post-processing cleaning approach not only augments (++) our own network architectures, RangeNet53, and Rangenet21, but also any other range-image-based method. See Figure 7.1 for an example.

7.1 Efficient Semantic Segmentation of LiDAR Point Clouds

The goal of our approach is to achieve accurate and fast semantic segmentation of point clouds, in order to enable autonomous machines to make decisions in a timely manner. To achieve this segmentation, we propose a projection-based 2D CNN processing of the input point clouds and utilize a range image representation of each laser scan to perform the semantic inference. We use in the following the term range image for the spherical projection of the point cloud, but each pixel, which corresponds to a horizontal and vertical direction, can store more than only a range value (e.g. can also hold coordinates, normals, remissions, etc.). The projection is followed by a fast, GPU-based, k-Nearest-Neighbor (kNN) search over the entire point cloud, which allows us to recover semantic labels for the entire input cloud. This is particularly critical when using small resolution range images since the projection would otherwise lead to a loss of information.

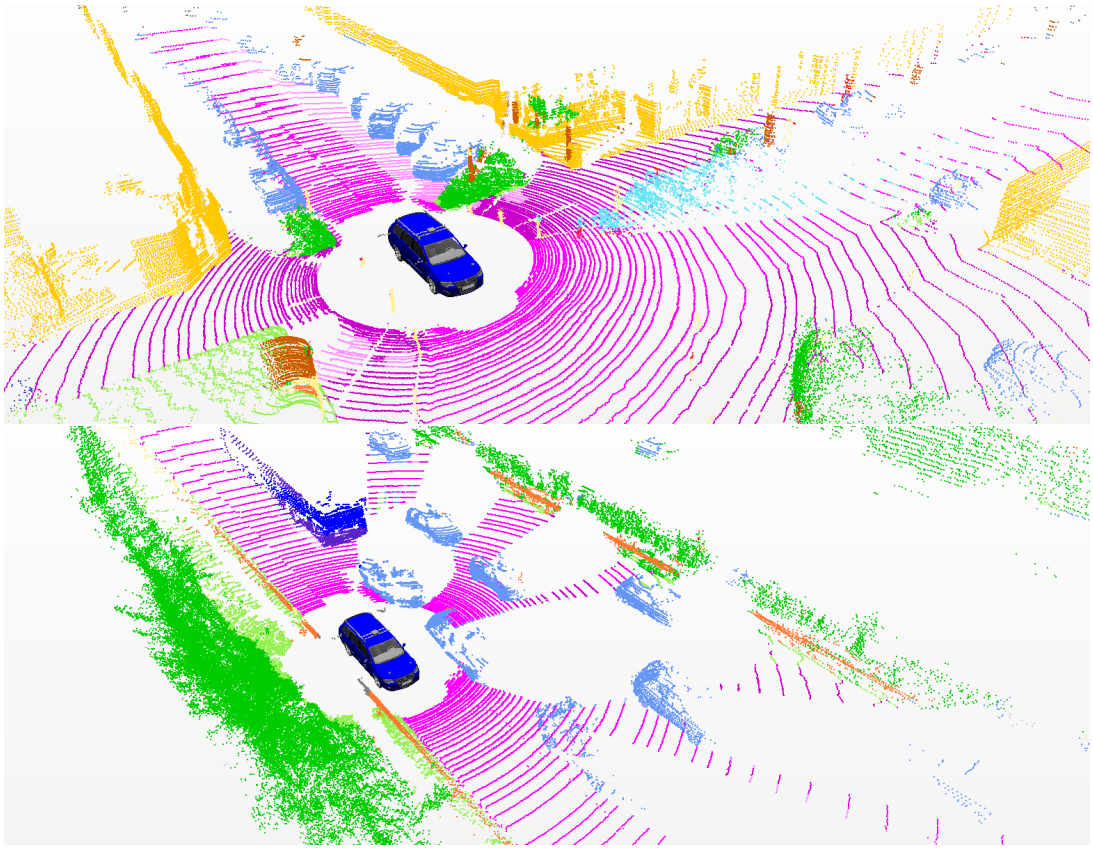

Figure 7.1: Velodyne HDL-64E laser scans from KITTI dataset hidden test set [46] with semantic information from our approach, RangeNet53++. Top: Sequence 13. Bottom: Sequence 20. Best viewed in color, each color represents a different semantic class

Our approach is divided into four steps, shown in Figure 7.2. These four steps are discussed in detail in the following subsections: (a) a transformation of the input point cloud into a range image representation, (b) a 2D fully convolutional semantic segmentation, (c) a semantic transfer from 2D to 3D that recovers all points from the original point cloud, regardless of the used range image discretization, and (d) an efficient range image-based 3D post-processing to clean the point cloud from undesired discretization and inference artifacts, using a fast, GPU-based kNN-search operating on all points.

In sum, we make three key claims: Our approach is able to (i) accurately perform semantic segmentation of LiDAR-only point clouds, surpassing the state of the art significantly, (ii) infer semantic labels for the complete original point cloud, avoiding to discard points regardless of the level of discretization used in the CNN, and (iii) work at the frame rate of a Velodyne scanner on an embedded computer that can easily fit in robots or in a vehicle.

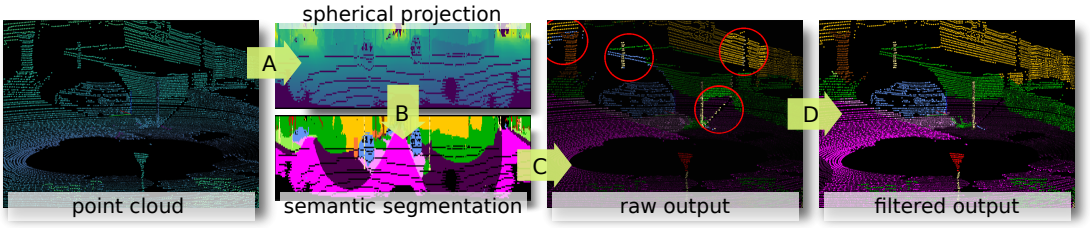

Figure 7.2: Block diagram of the approach. Each of the arrows corresponds to one of our modules.

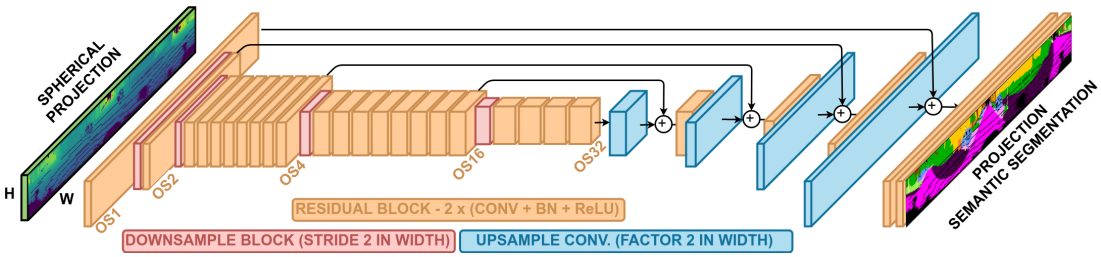

Figure 7.3: Our fully convolutional semantic segmentation architecture. RangeNet53 is inspired in a Darknet53 Backbone [144].

7.1.1 Range Image Point Cloud Proxy Representation

Several LiDAR sensors, such as the Velodyne sensor record the raw input data in a range image-like fashion. Each column represents the range measured by an array of laser range-finders at one point in time, and each row represents different turning positions for each of those range-finders, which are fired at a constant rate. However, on a vehicle moving at high speeds, this rotation does not happen fast enough to ignore the skewing generated by the vehicle motion. This leads to a sort of "rolling shutter" behavior more commonly observed in cameras. To obtain a more geometrically consistent representation of the environment for each scan, we must consider the vehicle motion, resulting in a point cloud which no longer contains range measurements for each pixel, but contains multiple measurements for some others. In order to obtain an accurate semantic segmentation of the full LiDAR point cloud, our first step is to convert each de-skewed point cloud into a range representation. For this, we convert each point $\mathbf{p}_i = (x, y, z)$ via a mapping $\Pi : \mathbb{R}^3 \mapsto \mathbb{R}^2$ to spherical coordinates and finally to image coordinates, as defined by

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\left[1 - \arctan(y, x)\,\pi^{-1}\right] & w \\ \left[1 - (\arcsin(z\,r^{-1}) + \mathrm{f}_{\mathrm{up}})\,\mathrm{f}^{-1}\right] & h \end{pmatrix}, \tag{7.1}$$

where (u, v) are said image coordinates, (h, w) are the height and width of the desired range image representation, $\mathrm{f} = \mathrm{f}_{\mathrm{up}} + \mathrm{f}_{\mathrm{down}}$ is the vertical field-of-view of the sensor, and $r = ||\mathbf{p}_i||_2$ is the range of each point, which is the Euclidean

distance to the origin of coordinates of the sensor. This procedure results in a list of (u, v) tuples containing a pair of image coordinates for each $\mathbf{p}_i$, which we use to generate our proxy representation. Using these indexes, we extract for each $\mathbf{p}_i$, its range r, its x, y, and z coordinates, and its remission, and we store them in the image, creating a $[5 \times h \times w]$ tensor. Because of the de-skewing of the scan, the assignment of each points to its corresponding (u, v) is done in descending range order, to ensure that all points rendered in the image are in the current field of view of the sensor. We furthermore save this list of (u, v) pairs to gather and clean the semantics of the resulting point cloud, as we describe in Section 7.1.3 and Section 7.1.4.

7.1.2 Fully Convolutional Semantic Segmentation

To obtain the semantic segmentation of this range image representation of the point cloud we use a 2D semantic segmentation CNN, which is modified to fit this particular input type and form factor. Similarly to Wu *et al.* [184], we use an encoder-decoder hour-glass-shaped architecture, which is depicted in Figure 7.3. These types of deep hour-glass-shaped segmentation networks are characterized by having an encoder with significant downsampling, which allows the higher abstraction deep kernels to encode context information while running faster than non-downsampling counterparts. In our case, this downsampling is 32 (see Figure 7.3). This is later followed by a decoder module which upsamples the "feature code" extracted by the convolutional backbone encoder to the original image resolution, adding also convolutional layers to refine these results. At the same time, after each upsampling we also add skip connections between different levels of output stride (OS) of the encoder and sum them to the corresponding output stride feature volume in the decoder, illustrated by the black arrows, to recover some of the high-frequency edge information that gets lost during the downsampling process. After this encoding-decoding behavior, the last layer of the architecture performs a set of $[1 \times 1]$ convolutions. This generates an output volume of $[n \times h \times w]$ logits, where n is the number of classes in our data. The last layer during inference is a softmax function over the unbounded logits of the form

$$\hat{y}_c = \frac{e^{\text{logit}_c}}{\sum_c e^{\text{logit}_c}} \tag{7.2}$$

This gives a probability distribution per pixel in the range image, where logit_c is the unbounded output in the slice corresponding to class c. During training, this network is optimized end to end using stochastic gradient descent and a

weighted cross-entropy loss $\mathcal{L}$:

$$\mathcal{L} = -\sum_{c=1}^{C} w_c \, y_c \, \log\left(\hat{y}_c\right), \tag{7.3}$$

$$\text{where the term } w_c = \frac{1}{\log\left(f_c + \epsilon\right)} \tag{7.4}$$

penalizes the class c according to the inverse of its frequency f_c. This handles imbalanced data, as is the case for most datasets in semantic segmentation, e.g., the class "road" represents a significantly larger number of points in the dataset than the class "pedestrian".

To extract rich features for our encoder backbone, we define our RangeNet architectures by modifying the Darknet [144] backbone architecture in a way that makes it usable for our purposes. This backbone was designed with general image classification and object detection tasks in mind and is very descriptive, achieving state-of-the-art performance in several benchmarks for these tasks. However, it was designed to work with square aspect ratio RGB images. The first necessary modification to the backbone is to allow the first channel to take images with 5 channels. As we are dealing with a sensor that has an array of 64 vertically-placed laser range-finders producing in the order of 130 000 points per scan, this leaves us with a range image of around $w = 2048$ pixels. To retain information in the vertical direction, we therefore only perform downsampling in the horizontal direction. This means that in the encoder, an OS of 32 means a reduction in w of a factor of 32, but 64 pixels still remain intact in vertical direction h. To evaluate how well our post-processing recovers the original point cloud information, in Section 7.2 we analyze input sizes of $[64 \times 2048]$, $[64 \times 1024]$, and $[64 \times 512]$, which produce feature volumes at the end of the encoder of size $[64 \times 64]$, $[64 \times 32]$, and $[64 \times 16]$ respectively.

7.1.3 Point Cloud Reconstruction from Range Image

The common practice to map from a range image representation to a point cloud is to use the range information, along with the pixel coordinates and the sensor intrinsic calibration to realize a mapping $\Pi^* : \mathbb{R}^2 \mapsto \mathbb{R}^3$. However, since we are generating the range image from a point cloud originally, as first introduced in Section 7.1.1, this could mean dropping a significant number of points from the original representation. This is especially critical when using smaller images in order to make the inference of the CNN faster. E.g., a scan with 130 000 points projected to a $[64 \times 512]$ range image will represent only 32 768 points, sampling the closest point in each pixel's frustum. Therefore, to infer all the original points in the semantic cloud representation, we use all the (u, v) pairs for all the points $\mathbf{p}_i$ obtained during the initial rendering process and index the range image with

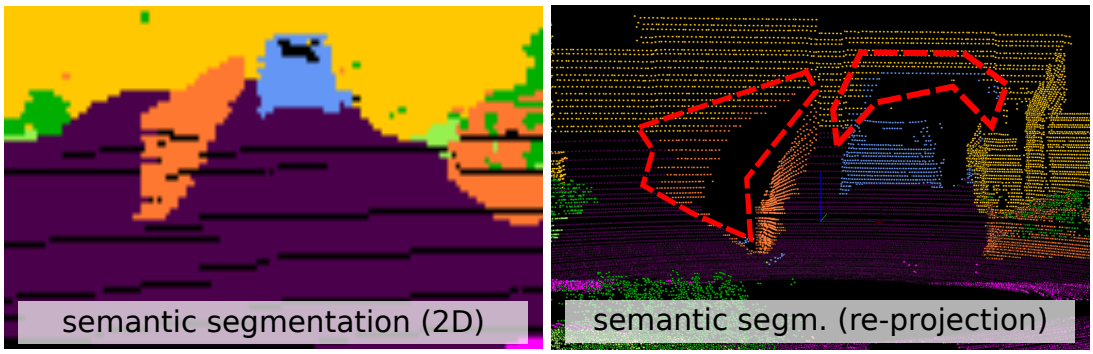

Figure 7.4: Illustration of the label re-projection problem. Both the fence and the car in the range image (left) were given the proper semantic label, but during the process of sending the semantics back to the original points (right), the labels were also projected as "shadows".

the image coordinates that correspond to each point. This can be performed extremely fast in the GPU before the next post-processing step takes place, and it results in a semantic label for each point that was present in the entire input scan, in a loss-less way.

7.1.4 Efficient Point Cloud Post-processing

Unfortunately, the benefits of the expedite semantic segmentation of LiDAR scans through 2D semantic segmentation of range images does not come without its drawbacks. The encoder-decoder hour-glass-like CNNs provide blurry outputs during inference, which is also a problem for RGB and RGBD semantic segmentation systems. Some methods, such as SqueezeSeg [184] use a conditional random field over the predictions in the image domain after the 2D segmentation to eliminate this "bleeding" of the output labels. Using the softmax probabilities of each pixel as unary potentials for the CRF, and penalizing jumps in signal and Euclidean distance between neighboring points. Even though this helps in 2D, it does not fix the problem after the re-projection to three-dimensional space, since once the labels are projected into the original point cloud, two or more points which were stored into the same range image pixel will get the same semantic label. This effect is illustrated in Figure 7.4, where the labels of the inferred point cloud present shadows in objects in the background due to the blurry CNN mask, and the mentioned discretization. Moreover, if we wish to use smaller range image representations to infer the semantics, this problem becomes even stronger, resulting in shadow-like artifacts of the semantic information in objects of different classes.

To solve this problem, we propose a fast, GPU-enabled, k-nearest neighbor (kNN) search operating directly in the input point cloud. This allows us to find, for each point in the semantic point cloud, a consensus vote of the k points in the scan that are the closest to it in 3D. As it is common in kNN search, we also

Algorithm 1: Efficient Projective Nearest Neighbor Search for Point Labels

Data: Range Image I_{range} of size $W \times H$,
Label Image I_{label} of predictions of size $W \times H$,
Ranges $\mathcal{R}$ for each point $\mathbf{p} \in \mathcal{P}$ of size N,
Image coordinates (u, v) of each point in R.
Result: Labels $L_{\text{consensus}}$ for each point of size N.

1 Let be range$lu = \{i \mid l \leq i \leq u\}$ the range from l to u.
/* Get S^2 neighbors N' for each pixel */
2 **foreach** $(u, v) \in$ range$1W \times$ range$1H$ **do**
3 **foreach** $(i, j) \in$ range$1S \times$ range$1S$ **do**
4 $N'[v \cdot W + u, j \cdot S + i] = I_{\text{range}}[u + i, v + j]$
5 **end**
6 **end**
/* Get neighbors N for each point */
7 **foreach** $(u, v) \in \mathcal{C}$ **do**
8 **foreach** $(i, j) \in$ range$1S \times$ range$1S$ **do**
9 $N[v \cdot W + u, i \cdot S + j] = N'[v \cdot W + u, i \cdot S + j]$
10 **end**
11 **end**
/* Fill in real point ranges */
12 **foreach** $i \in$ range$1N$ **do**
13 $N[i, \lfloor (S \cdot S - 1)/2 \rfloor] = \mathcal{R}(i)$
14 **end**
/* Label neighbors L' for each pixel */
15 **foreach** $u \in$ range$1W$, $v \in$ range$1H$ **do**
16 **foreach** $(i, j) \in$ range$1S \times$range$1S$ **do**
17 $L'[v \cdot W + u, i \cdot S + j] = I_{\text{label}}[u + i, v + j]$
18 **end**
19 **end**
/* Get label neighbors L for each point */
20 **foreach** $(u, v) \in \mathcal{C}$ **do**
21 **foreach** $(i, j) \in$ range$1S \times$range$1S$ **do**
22 $L[v \cdot W + u, i \cdot S + j] = L'[v \cdot W + u, i \cdot S + j]$
23 **end**
24 **end**
/* Distances to neighbors D for each point */
25 **foreach** $i \in n$range$1N$ **do**
26 **foreach** $j \in n$range$1S \cdot S$ **do**
27 $D[i, j] = |N[i, j] - \mathcal{R}(i)|$
28 **end**
29 **end**
/* Compute inverse Gaussian Kernel */
30 Let $\mathcal{N}(u \mid \mu, \sigma)$ be a Gaussian with mean μ and std. deviation σ.
31 **foreach** $(i, j) \in$ range$1S \times$range$1S$ **do**
32 $G'[j \cdot S + i] = \mathcal{N}(i \mid 0, \sigma) \cdot \mathcal{N}(j \mid 0, \sigma)$
33 **end**
34 Let be $G_{\max} = \max \{G'[i] \mid i \in$ range$1S \cdot S\}$ the maximium of G'
35 **foreach** $i \in$ range$1S \cdot S$ **do**
36 $G[i] = 1 - G_{\max} \cdot G'[i]$
37 **end**
/* Weight neighbors with inverse Gaussian kernel */
38 **foreach** $i \in$ range$1N$ **do**
39 **foreach** $j \in$ range$1S \cdot S$ **do**
40 $D[i, j] = D[i, j] \cdot G[j]$
41 **end**
42 **end**
/* Find k-nearest neighbors $\mathcal{S}$ for each point */
43 **foreach** $i \in$ range$1N$ **do**
44 $\mathcal{S}[i] = \{j \mid |\{n \in$ range$1S \cdot S \mid D[i, n] < D[i, j]\}| \leq k\}$
45 **end**
/* Gather votes. */
46 **foreach** $i \in$ range$1N$ **do**
47 $n = 1$
48 **foreach** $j \in \mathcal{S}[i]$ **do**
49 **if** $D[i, j] > \delta_{cutoff}$ **then**
50 $L_{\text{knn}}[i, n] = C + 1$
51 **end**
52 **else**
53 $L_{\text{knn}}[i, n] = L[i, j]$
54 **end**
55 $n = n + 1$
56 **end**
57 **end**
58 **foreach** $j \in$ range$[1 : k]$ **do**
59 **if** $L_{knn}[i, j] \leq C$ **then**
60 $V[i, L_{\text{knn}}[i, j]] = V[i, L_{\text{knn}}[i, j]] + 1$
61 **end**
62 **end**
/* Find maximum consensus. */
63 **foreach** $i \in$ range$1N$ **do**
64 $L_{\text{consensus}}[i] = \arg\max_c L_{\text{knn}}[i, c]$
65 **end**

set a threshold for the search, which we call *cutoff*, setting the maximum allowed distance of a point considered a near neighbor. The distance metric to rank the k closest points can be the absolute differences in the range or the Euclidean distance. Note that we also tried to use the remission as a penalization term, but it did not help in our experience. For the remainder of this section, we explain the approach considering the usage of the absolute range difference as the distance, but the Euclidean distance works analogously, albeit being slower to compute.

We explain the steps of our algorithm, described in Algorithm 1, referring to the corresponding line numbers. Our approximate nearest neighbor search uses the range image representation to obtain, for each point in the $[h, w]$ range image, an $[S, S]$ window around it in the image representation, with S being a value found empirically through a grid-search in the validation set. This operation is performed through the *"im2col"* primitive, which is internally used by most parallel computing libraries to calculate a convolution, and therefore di-

rectly accessible through all deep learning frameworks. This results in a matrix of dimension $[S^2, hw]$, which contains an unwrapped version of the $[S, S]$ neighborhood in each column, and each column center contains the actual pixel's range (lines 2–4). As not all points are represented in the range image, we use the (u, v) tuples for each $\mathbf{p}_i$ obtained during the range image rendering process, and extend this representation to a matrix of dimension $[S^2, N]$, containing the range neighborhoods of all the scan points (lines 5–7). As this is done by indexing the unfolded image matrix, the centers of columns do not represent the actual range values anymore. Thus, we replace the center row of the matrix by the actual range readings for each point. The result of this is a $[S^2, N]$ matrix which contains all the range readings for the points in the center row, and in each column, its unwrapped $[S, S]$ neighborhood (lines 8–9). This is a key checkpoint in the algorithm because it allows us to find in a quick manner a set of S^2 candidates to consider during the neighbor search for each point, in parallel. This allows our algorithm to run orders of magnitude faster than other nearest neighbor search approaches such as the popular FLANN methods [117], which work in unordered point clouds, by exploiting the arrangement of the scan points in the sensor. This key structural difference allows us to run in real-time even for large point clouds.

The following two steps are analogous to this unwrapping (lines 10–15), but instead of obtaining the ranges of the neighbor candidates, it contains their labels. This $[S^2, N]$ label matrix is later used to gather the labels for the consensus voting, once the indexes for the k neighbors are found. At this point in the algorithm, we are able to calculate the distance to the actual point for each of the S^2 candidates. If we subtract the $[1, N]$ range representation of the LiDAR scan from each row of the $[S^2, N]$ neighbor matrix, and point-wise apply the absolute value, we obtain a $[S^2, N]$ matrix where each point contains the range difference between the center of the neighborhood, which is the query point, and the surrounding points (lines 16–18). The fact that we are using a small $[S, S]$ neighborhood search around each point allows us to make the assumption that the absolute difference in the range is a good proxy for the euclidean distance, since points that are close in (u, v) coordinates will only have a similar range if their actual distance in 3D space is similar. This is tested empirically in our experimental section, allowing us to make the distance calculation more efficient, and obtaining the same result for the post-processing.

The next step is to weight the distances by an inverse Gaussian kernel, which penalizes the bigger differences in the range between points that are distant in (u, v) more. This is done by the point-wise multiplication of each column with the unwrapped kernel (lines 19–27).

After this, we need to find the k closest points for each column containing the S^2 candidates, which is done through an "argmin" operation (lines 28–29). This

allows us to get the indexes for the k points in the S neighborhood with the least weighted distance.

The last step in our search is to check which ones of those k points fit within the allowed threshold, which we called *cutoff*, and accumulate the votes from all the labels of the points within that radius. This is performed through a "gather add" operation, which generates a $[C, N]$ matrix, where C is the number of classes, and each row contains the number of votes in its index class (lines 30–41). A simple "argmax" operation over the columns of this matrix returns a $[1, N]$ vector, which contains the clean labels for each point in the input LiDAR point cloud, and serves as the output of our approach (lines 42–43).

It is important to notice that, given the independence of all the points inside the loops in Algorithm 1, each of the main components can be represented either with a parallel computing primitive or in a highly vectorized way, both of which are directly implementable in a GPU, using off-the-shelf, high abstraction deep learning or data science frameworks.

This algorithm requires setting four different hyperparameters: (i) S which the size of the search window, (ii) k which is the number of nearest neighbors, (iii) *cutoff* which is the maximum allowed range difference for the k, and (iv) σ, which is the standard deviation for the inverse Gaussian. The values for the hyperparameters are calculated empirically through a data-driven search in the validation set of our training data, and a brief analysis is provided in the experimental section.

7.2 Experimental Evaluation

The experimental evaluation is designed to evaluate if our claims about our approach are valid. These claims are that our approach: (i) outperforms the state of the art in the task of semantic segmentation of LiDAR scans, (ii) infers the entire point cloud while recovering the high-frequency information in the un-projection step, and (iii) runs online in an embedded computer at sensor frame-rate.

Dataset. We train and evaluate our approach on our large-scale dataset described in Chapter 6, which provides dense point-wise annotations for the entire KITTI Odometry Benchmark [9, 46]. The dataset is comprised of over 43 000 scans from which over 21 000 from sequences 00 to 10 are available for training and the remaining scans from sequences 11 to 21 are used as the test set. We used sequence 08 as the validation set for hyperparameter selection and trained our approach on the remaining training sequences. Overall, the dataset provides 22 classes from which 19 classes are evaluated on the test set via our benchmark website.

Hyperparameter selection. All hyperparameters for RangeNet models are selected and evaluated on the validation set (sequence 8). For all backbone training, we use a learning rate of 0.001, with a decay of 0.99 every epoch, and train for 150 epochs. For all CNN backbones, convergence was achieved in less than 150 epochs. For all the state-of-the-art methods, the hyperparameters were also selected on the validation set.

Metrics. To assess the labeling performance, we use the commonly applied mean Jaccard Index or mean intersection-over-union (IoU) metric, mIoU, over all classes [40] given by Equation (6.1).

To better assess the performance with respect to the precision of the prediction, we propose an additional evaluation metric which we call *border-IoU*. This metric is defined in the same way as the standard IoU, but only applies within the subset of points defined by an extra parameter, which considers how far a point is to the self-occlusion of the sensor, which is manifested in a change in the label in the range image. This metric is designed to show how much our algorithm can help the "shadow-like" wrong label projections in the semantic point clouds.

7.2.1 Performance of RangeNet++ in Comparison to the State of the Art

The first experiment is designed to support our claim that our approach overperforms the state of the art in the task of scene semantic segmentation of LiDAR point clouds. Table 7.1 shows the difference between our RangeNet backbones, using 21 and 53 layers (RangeNet21 and RangeNet53, respectively), and 13 other baseline methods. Note that all approaches that surpass the performance of our approach [114], post-date our work, and many of them [3, 187, 31] are based on it. Figure 7.5 shows a visual comparison of the projected results of RangeNet++ with the state of the art at the time this work was published in November 2019 [114].

The superior performance of our RangeNet baselines, even without our cleaning, for all the input resolutions of shows that it is a solid baseline to benchmark our efficient kNN cleaning. Table 7.2 also shows that our method, RangeNet++, which includes our kNN post-processing consistently beats its unprocessed RangeNet counterpart, showing the efficacy of our kNN search. The kNN cleaning is consistently better for all but one class, unlike the CRF, which is a conclusion reached by the original SqueezeSeg [184] work as well, even when the overall IoU is higher.

Table 7.1: IoU [%] on test set (sequences 11 to 21). RangeNet21 and RangeNet53 represent the new baselines with augmented Darknet backbones (21 and 53 respectively), and the versions with (++) are treated with our fast point cloud post-processing based on range. Methods marked with * post-date our work by at least 6 months.

Approach	Size	meanIoU	Scans/sec
Pointnet [139]	50K pts	14.6	2
Pointnet++ [141]		20.1	0.1
SPGraph [81]		20.0	0.2
SPLATNet [166]		22.8	1
TangentConv [171]		35.9	0.3
SqueezeSeg [184]	64 × 2048px	29.5	**66**
SqueezeSeg-CRF [184]		30.8	55
SqueezeSegV2 [185]		39.7	50
SqueezeSegV2-CRF [185]		39.6	40
RangeNet21[Ours]		47.4	20
RangeNet53 [Ours]	64 × 2048px	49.9	13
	64 × 1024px	45.4	25
	64 × 512px	39.3	52
RangeNet53++ [Ours+kNN]	64 × 2048px	**52.2**	12
	64 × 1024px	48.0	21
	64 × 512px	41.9	38
RandLANet* [65]	50K pts	53.9	21
LatticeNet* [154]	*all* pts	52.9	7
PolarNet* [193]	variable	54.3	16
SqueezeSegV3* [187]	64 × 2048px	55.9	6
SalsaNext* [31]	64 × 2048px	54.5	**83**
3DMininet* [3]	64 × 2048px	55.8	28
KPConv* [174]	variable	**58.8**	3

Table 7.2: IoU [%] on test set (sequences 11 to 21) for both unprocessed and postprocessed RangeNet53 models. We show that regardless of the resolution, the post-processed counterpart consistently outperforms the raw output of the segmentation CNN.

Input Size	KNN	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	motorcyclist	road	parking
64 × 512 px	✗	81.0	9.9	11.7	19.3	7.9	16.8	25.8	2.5	90.1	49.9
	✓	87.4	9.9	12.4	19.6	7.9	18.1	29.5	2.5	90.0	50.7
64 × 1024 px	✗	84.6	20.0	25.3	24.8	17.3	27.5	27.7	7.1	90.4	51.8
	✓	90.3	20.6	27.1	25.2	17.6	29.6	34.2	7.1	90.4	52.3
64 × 2048 px	✗	86.4	24.5	32.7	25.5	22.6	36.2	33.6	4.7	**91.8**	64.8
	✓	**91.4**	**25.7**	**34.4**	**25.7**	**23.0**	**38.3**	**38.8**	**4.8**	**91.8**	**65.0**

Input Size	KNN	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign	mean IoU
64 × 512 px	✗	69.4	2.0	76.0	45.5	74.2	38.8	62.7	25.5	38.1	39.3
	✓	70.0	2.0	80.2	48.9	77.1	45.7	64.1	37.1	42.0	41.9
64 × 1024 px	✗	72.1	22.8	80.4	50.0	75.1	46.0	62.7	33.4	43.4	45.4
	✓	72.7	22.8	83.9	53.3	77.7	52.5	63.7	43.8	47.2	48.0
64 × 2048 px	✗	74.6	**27.9**	84.1	55.0	78.3	50.1	64.0	38.9	52.2	49.9
	✓	**75.2**	27.8	**87.4**	**58.6**	**80.5**	**55.1**	**64.6**	**47.9**	**55.9**	**52.2**

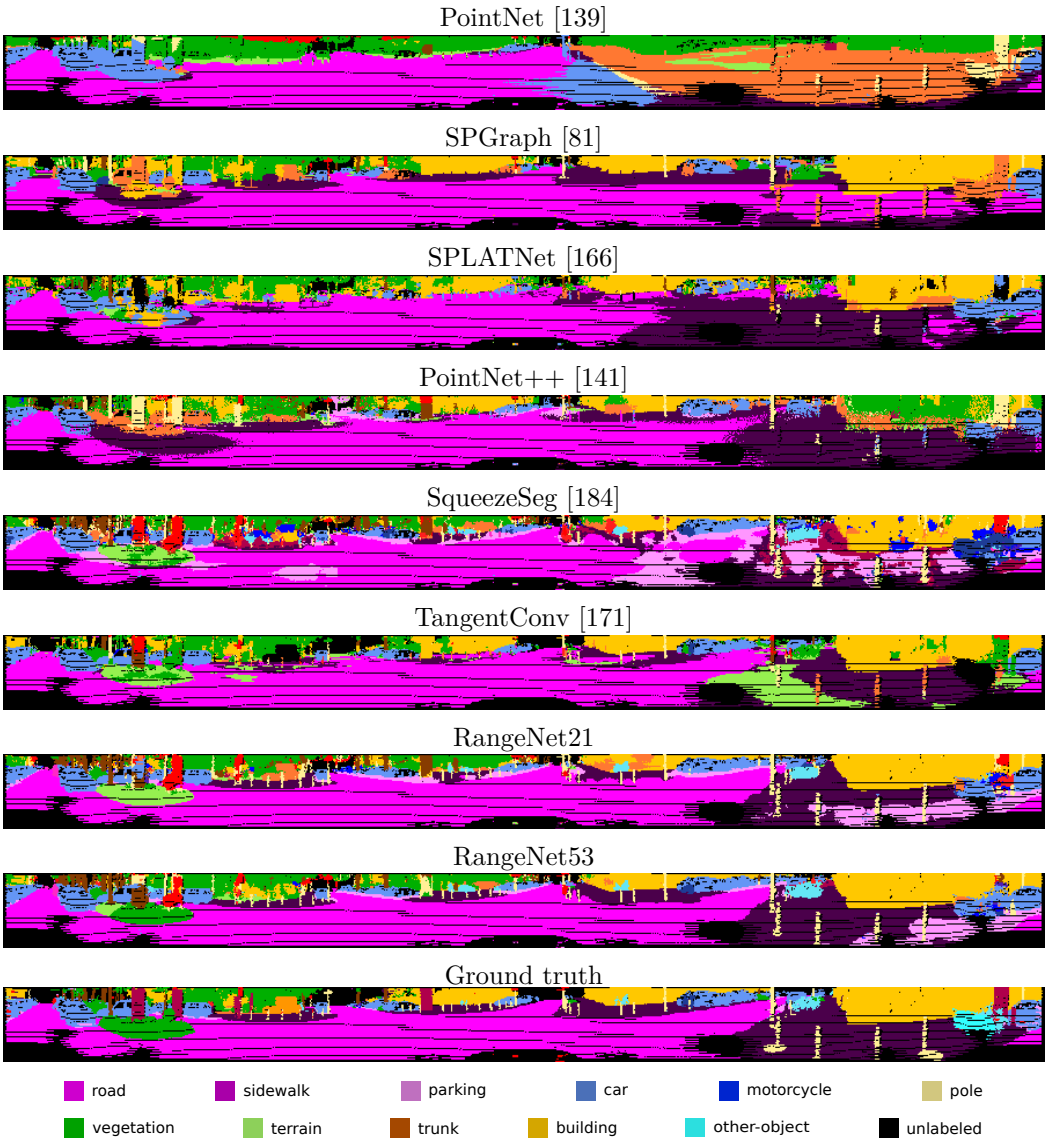

Figure 7.5: Examples of inference for all methods. The point clouds were projected to 2D using a spherical projection to make the comparison easier. All images show results on the same scan from the validation set. Best viewed in color.

7.2.2 Filter Parameter Influence

The second experiment shows the influence of the k and S parameters in the validation set. For each of the 4 parameters k, S, σ, and *cutoff* we chose a wide range of values and evaluated the result of post-processing the inference results of the RangeNet53 backbones for all input resolutions. Figure 7.6 shows a normalized result of the IoU in the validation set for each parameter set, for various k and S and the "argmax" of σ and *cutoff*. The results also show that we can obtain similar results using small kernels and the absolute range difference, as a proxy for the Euclidean distance. This supports our statement that range difference is a good proxy for the actual distance if points in the image are close.

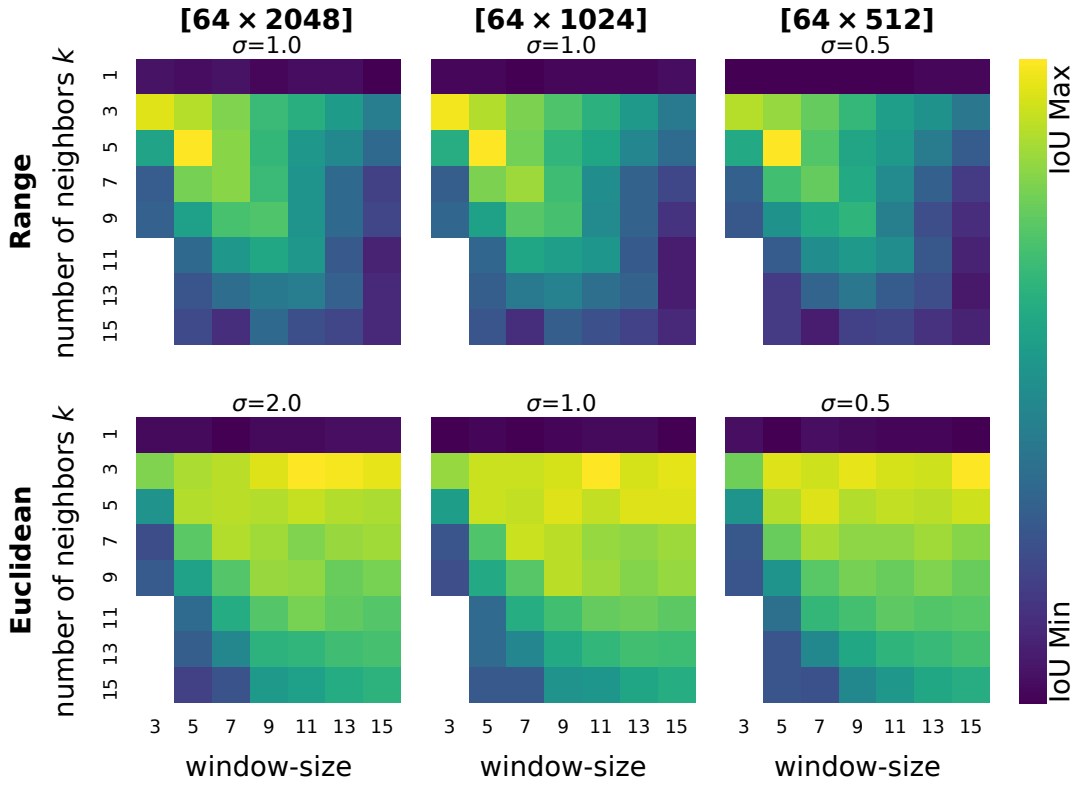

Figure 7.6: Hyperparameter search post-processing for both range (top row) and Euclidean distance (bottom row) using RangeNet53++, and different input resolutions. All experiments used $cutoff = 1.0\,\text{m}$.

7.2.3 Post-Processing Influence

The third experiment is designed to support investigate if our algorithm improves the reconstruction of the semantics of the entire point cloud even for smaller range image resolutions. For this, we use our *border-IoU* metric, which only considers points that are a certain number of points away from a change in label. In Figure 7.7 we show the value of the IoU and the value of the border IoU for different distances to the border. Note that our post-processing approach does not only improve the IoU score by a couple of % points, but it significantly improves the border IoU score for low values of the distance to the border parameter. This means that our approach is especially useful to help in cases of label "bleeding" or "shadowing" described in Section 7.1.4. Another important conclusion is that there are only marginal differences between using the faster to compute range difference and the actual Euclidean distance, through-out the entire spectrum of border distances, and in the IoU, which support our statement that it is a good approximation.

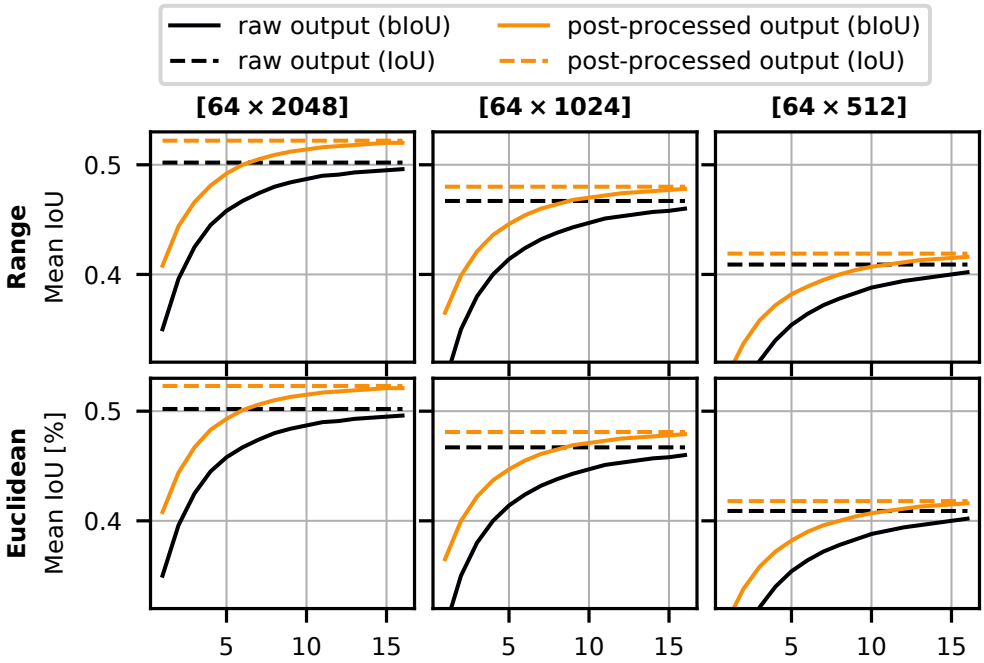

Figure 7.7: Border IoU (bIoU) and IoU as a function of the distance to label change. This plot shows that our post-processing improves the IoU, and significantly improves the borderIoU, which means that it recovers blurry mask and discretization errors better.

7.2.4 Runtime

The fourth experiment is designed to show that the approach can run in its totality online in a moving platform, using a single GPU at the frame rate of the laser scanner. Table 7.3 shows the runtime for the backbone, different post-processing distance functions (for the best parameters), and the total time required. As expected, the range-based post-processing is faster to calculate, since each distance calculation requires a subtraction and an absolute value, compared to 3 squares, 2 sums, and 1 square root. Therefore, since the difference in performance is negligible, we use the sum of our CNN backbone plus this range processing time for our total runtime, which we evaluate in two different types of hardware.

Table 7.3: Runtime of RangeNet53++. Sensor frame rate is 10FPS.

Hardware	Resolution (px)	Processing time (ms) CNN	Range	Euclid	Total	FPS
Quadro P6000	64 × 512	19	7	11	26	38
	64 × 1024	40			47	21
	64 × 2048	75			82	12
Jetson AGX	64 × 512	45	35	52	80	13
	64 × 1024	87			122	8
	64 × 2048	153			188	5

7.3 Related Work

Semantic segmentation for autonomous driving using images made an immense progress in recent years due to the advent of deep learning and the availability of increasingly large-scale datasets for the task, such as CamVid [17], Cityscapes [30], or Mapillary [119]. Together, this enables the generation of complex deep neural network architectures with millions of parameters achieving high-quality results. Prominent examples are Deeplab V3 [23] and PSPNet [195].

Despite their impressive results, these architectures are too computationally expensive to run in real-time on an autonomous system, which is a must for autonomous navigation exploiting semantic cues. This spawned the creation of more efficient approaches such as Bonnet [113], ENet [126], ERFNet [151], and Mobilenets V2 [159], which leverage the law of diminishing returns to find the best trade-off between runtime, the number of parameters, and accuracy. These, however, are designed for images and not for LiDAR scans.

Transferring these results to LiDAR data has, so far, been hindered by two factors: (i) the lack of publicly available large-scale datasets for the task of semantic segmentation in autonomous driving and (ii) how prohibitively expensive to run most LiDAR semantic segmentation models are.

To tackle the problem of the lack of data, Wu *et al.* [184, 185] used the provided bounding box of the KITTI dataset [46]. They also leveraged simulation to generate realistic-looking scans from a game engine. We have released the first large-scale dataset with full semantic segmentation of LiDAR scans [9], in which all scans of the KITTI odometry dataset [46] were densely annotated, i.e., over 43 000 scans, with over 3.5 billion annotated points. Without the data-starvation barrier, this work investigates which of the current state-of-the-art algorithms can be exploited and adapted for point cloud in the autonomous driving context.

Leveraging large datasets for other contexts [32, 55], several deep learning-based methods for 3D semantic segmentation were recently developed, such as PointNet [139], PointNet++ [141], TangentConvolutions [171], SPLATNet [166], SuperPointGraph [81], and SqueezeSeg [184, 185].

One of the problems of dealing with point cloud data directly is the lack of a proper ordering, which makes learning order-invariant feature extractors extremely challenging. Qi *et al.* [139, 141] use as inputs the raw, un-ordered point clouds and apply symmetrical operators that are able to deal with this ordering problem. For this purpose, max pooling is used by PointNet [139] to combine the features and generate permutation-invariant feature extractors. This, however, is a limiting factor for PointNet, causing it to lose the ability to capture spatial relationships between features. This limits its applicability to complex scenes. PointNet++ [141] tackles this problem by using a hierarchical approach for fea-

ture extraction. By exploiting individual PointNets in a local vicinity, it captures short-range dependencies and then applies this concept hierarchically to capture global dependencies.

Tatarchencko *et al.* [171] take a different approach to handle unstructured point clouds. They propose TangentConvolutions that apply CNNs directly on surfaces, which can only be achieved if neighboring points are sampled from the same surface. In this case, the authors can define a tangent convolution as a planar convolution that is applied to the projection of the surface at each point. This assumption is, however, violated in case of a rotating LiDAR and the generated distance-dependent sparsity of the point cloud.

Su *et al.* [166] approach the representational problem differently in SPLATNet, by projecting the points in a high-dimensional sparse lattice. However, this approach does not scale well both in terms of computation and memory consumption. To alleviate this, bilateral convolutions [71] allow them to apply theses operators exclusively on occupied sectors of the lattice.

Landrieu *et al.* [81] manage to summarize the local relationships in a similar fashion to PointNets by defining a SuperPoint Graph. This is achieved by creating so-called SuperPoints, which are locally coherent, geometrically homogeneous groups of points that get embedded by a PointNet. They create a graph of SuperPoints that is an augmented version of the original point cloud and train a graph convolutional network to encode the global relationships.

In the case of rotating LiDAR segmentation, the number of points per scan is in the order of 10^5. This scale prevents all of these aforementioned methods from running in real-time, limiting their applicability in autonomous driving. In contrast, we propose a system that provides accurate semantic segmentation results, while still running at frame-rate of the sensor.

Leading the charge in online processing, SqueezeSeg and SqueezeSegV2, by Wu *et al.* [184, 185], also use a spherical projection of the point cloud, enabling the usage of 2D convolutions. Furthermore, a light-weight fully convolutional semantic segmentation is applied along with a conditional random field (CRF) to smooth the results. The last step is an un-discretization of the points from the range image back into the 3D world. Both are capable of running faster than the sensor rate, i.e., 10 Hz, and we use them as the basis of our approach.

Several limitations needed to be addressed in order to provide full semantic segmentation with this framework. First, the projection needed to be extended to include the full field-of-view of the LiDAR scan, since the SqueezeSeg framework only uses the frontal 90 degrees of the scan, where the objects of the original KITTI dataset labels are annotated by bounding boxes. Second, the SqueezeNet backbone is not descriptive enough to infer all the 19 semantic classes provided by our dataset [9]. Third, the CRF needed to be replaced, by our efficient, GPU-

based nearest neighbor search acting directly on the full, unordered point cloud. This last step enables the retrieval of labels for all points in the cloud, even if they are not directly represented in the range image, regardless of the used resolution.

7.4 Conclusion

In this work, we presented a fast and accurate framework for semantic segmentation of point clouds recorded by a rotating LiDAR sensor. Our main contribution is a novel deep-learning-supported approach that exploits range images and 2D convolutions, followed by a novel, GPU-accelerated post-processing to recover consistent semantic information during inference for entire LiDAR scans. Our experimental evaluation suggests that our modified 2D deep CNN operating on range images outperforms the current state of the art in semantic segmentation of LiDAR point clouds. Moreover, our efficient, GPU-enabled post-processing can further improve on these results by recovering important boundary information lost during the de-skewing of the laser scans, the lossy discretization into a proxy representation, and the inference through an hour-glass-shaped CNN. Overall, our approach outperforms the state of the art both in accuracy and runtime, taking a step forward towards sensor redundancy for semantic segmentation for autonomous vehicles and robots. Since the publication of our benchmark presented in Chapter 6, as well as the work presented in this chapter [114], a slew of other works using raw point clouds as well as projective methods have been developed [65, 154, 193, 187, 31, 3, 174]. All of them fall into one of three categories (or all): they find different ways to project the data into the proxy representation, they use a different loss function that is a better proxy for the task of semantic segmentation, or they make architectural changes to exploit the variable sparsity of the data in their favor. Regardless of the category, these are all signals that indeed, the development of novel methods to semantic segmentation of LiDAR-only point clouds were stagnant due to the lack of openly available large-scale labeled data. Furthermore, the fact that some works are already basing their improvements on this work [3, 187, 31] highlights the usability of the approach, as well as the ease of use of the open-source library where we released it: `http://www.github.com/PRBonn/lidar-bonnetal`.

Chapter 8

LiDAR Panoptic Segmentation

As highlighted by the last two chapters, perception and scene understanding are key components for building fully autonomous cars that can drive safely even in unknown parts of the world. A multitude of sensors such as cameras, LiDARs, and radars offering redundant views of the world are part of the hardware stack of these vehicles. Image-based perception has been steadily becoming more capable due to advances in deep learning [85] and convolutional neural networks. LiDARs are often used sensors because they produce accurate distance measurements, even in scenarios where other sensors fail, like at night.1 However, challenges caused by the characteristics of the LiDAR data, such as its distance-dependent sparsity, consequently call for different solutions.

Panoptic segmentation [76] is a recently proposed task unifying semantic segmentation of so-called *stuff* classes and instance-specific *thing* classes jointly. Specifically, *stuff* refers to uncountable classes, such as *vegetation*, or *road*, but also countable classes that are not critical to distinguish individually when performing a specific task, such as is the case for *buildings* while driving. Opposite to this, *things* represent interesting countable classes for the task the robot performs, such as driving participants (i.e., *cars*, *pedestrians*, etc). Therefore, this task provides a unified understanding of the scene components, leading towards scene understanding capturing the complete picture. See Figure 8.1 for an illustration.

Recently, many approaches to this task using images were proposed [76, 89, 93, 115, 134, 186]. In this work, we investigate the task of panoptic segmentation using solely LiDAR scans and present an approach for solving this task, as shown in Figure 8.1. First, we propose using state-of-the-art methods to *independently* solve the tasks of semantic segmentation and object detection providing 3D bounding boxes, and merging the predictions. This approach is computationally too expensive and thus not suited for real-world deployment, but provides a strong baseline for an online approach. Then, we propose a single-stage ap-

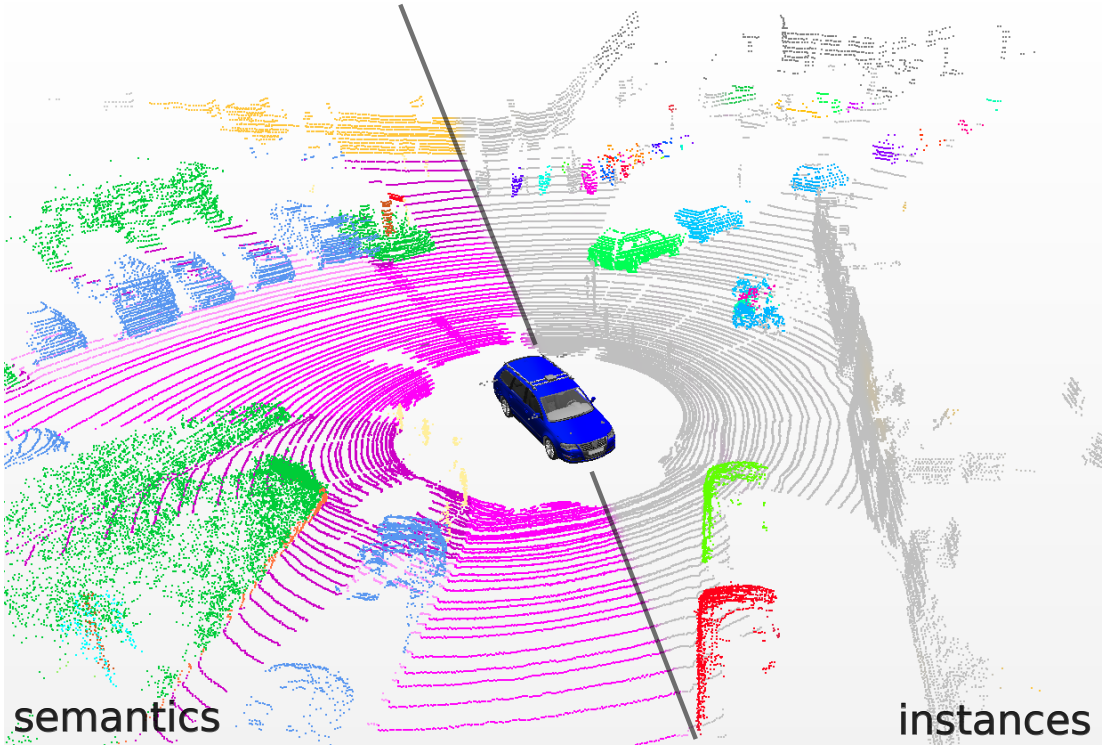

Figure 8.1: Our approach provides a panoptic segmentation for point clouds from a rotating automotive LiDAR. Thus, we assign each point a semantic label (right part), and instance IDs (left part).

proach that jointly solves the semantic and instance segmentation of *things*, as well as the semantic segmentation of *stuff*, which comprises the task of panoptic segmentation.

8.1 Efficient Panoptic Segmentation of LiDAR Point Clouds

Similarly to Chapter 5, we propose a single-stage architecture that learns both instance and semantic embeddings using a shared encoder, in order to jointly infer panoptic segmentation labels. The instance decoder provides an offset prediction for each point that points towards the object center, which allows us to segment individual instances. In turn, the semantic embeddings are used to extract point-wise semantic labels and aid the clustering of object centers. Furthermore, we propose a novel upsampling method that allows us to use large output strides, enabling better runtime performance. Combined with a category-based loss, we achieve high panoptic quality for the panoptic task [11] of SemanticKITTI [9] than an approach combining state-of-the-art projective semantic segmentation [114] and state-of-the-art object detection [82]. Lastly, our experiments show that

our single-stage approach runs at a fraction of the time compared to two two-stage approaches, which is paramount for moving vehicles. In summary, our contributions are: (i) a single-stage approach for LiDAR panoptic segmentation that achieves state-of-the-art performance at a fraction of the processing time of two-stage approaches, (ii) a novel upsampling strategy exploiting the distance information provided by the LiDAR point clouds leading to better panoptic quality and increased runtime efficiency, and (iii) the novel combination of semantic and geometric embeddings with learned point-wise radii for metric learning-based instance clustering.

8.1.1 Baseline Two-Stage Approaches

Since panoptic segmentation for LiDAR point clouds in the context of autonomous driving is a novel task which is made possible by the data presented in Chapter 6, we need to find strong baselines in the literature that can be adapted to compare against our novel, single-stage approach.

To this end, we propose a combination of state-of-the-art semantic segmentation approaches on SemanticKITTI, KPConv [174] and RangeNet++ [114], with a state-of-the-art object detector, namely PointPillars [82] to provide instance-level information. To extract the instance information, we use the oriented bounding boxes of the object detector, i.e., bounding boxes for *cars*, *pedestrians*, and *cyclists* trained on the KITTI detections benchmark [46], and extract the instance ID for points inside the bounding boxes. By combining the predictions of the semantic segmentation, and assigning the instance ID of each bounding box to each point inside of it, we obtain a panoptic segmentation. Note that we only assign instance IDs to points from the *thing* classes, e.g., a *car* point classified as *road* or *parking* is not assigned an ID.

We used pre-trained models or publicly available predictions for KPConv [174] and RangeNet++ [114], which were trained on SemanticKITTI. PointPillars had to be trained from scratch using the provided implementation[1], modifying the configuration of the object detector such that it provides region proposals and bounding boxes for the full 360-degree field-of-view of the LiDAR sensor. These networks were run independently for semantic segmentation and object detection and then merged to generate a panoptic segmentation. Neither approach can, therefore, run at the frame rate of the LiDAR, i.e., 10 Hz, having computational budgets that are not suitable in a self-driving car. Furthermore, the PointPillars detector [82] requires training separate networks, one for the class *car* and one for *pedestrian* combined with *cyclist*, which accentuates the problem further. We provide an evaluation of the performance and runtime in our experimental section.

[1] See the github repository at `https://git.io/Je25l`.

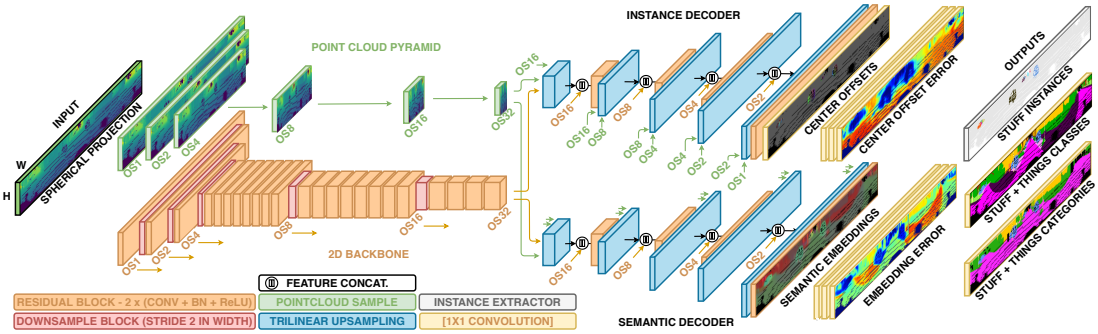

Figure 8.2: Architecture layout for the single-stage, projective panoptic segmentation approach. For detailed samples of inputs, intermediate representations, and outputs see Figure 8.3

To move a step closer to a multi-task network that can run in real-time on a self-driving car, we propose a single-stage approach based on RangeNet++ [114], which is not the best performing approach on the semantic segmentation task, but was the best real-time at the time this work was created and submitted, in February 2020.

8.1.2 Proposed Single-Stage Approach

We present a novel, single-stage, and real-time capable panoptic segmentation approach using a shared encoder with a semantic and instance decoder. We leverage the geometric information of the LiDAR scan to perform a novel, distance-aware tri-linear upsampling, which allows our approach to use larger output strides than using transpose convolutions leading to substantial savings in computation time. Our experimental evaluation and ablation studies for each module show that combining our geometric and semantic embeddings with our learned, variable instance thresholds, a category-specific loss, and the novel trilinear upsampling module leads to higher panoptic quality.

Figure 8.2 illustrates our network architecture. First, the point cloud obtained by the LiDAR scanner is projected to a range-image-like representation containing the range, (x, y, z) point coordinates, and remission of each point by virtue of a spherical projection of the de-skewed scans caused by the ego-motion of the vehicle. Then, we extract features at different resolutions, or output strides (OS), using a shared backbone, which is trained with all the losses through backpropagation. At the same time, we construct a point cloud pyramid which samples points from the image representation of the latter at exactly the location where the downsampling, stride 2 convolutions are applied in the backbone. This helps us recover the features with higher accuracy during the upsampling process.

Following the backbone and image pyramid, we use two separate decoders, bringing back the backbone features to the original image resolution, which also contain convolutional layers that learn task-specific functions. The first decoder

extracts a semantic embedding $\widehat{\mathbf{e}}_p$ that allows us to predict classes and categories, as well as an error estimate for the embeddings, used in the clustering. The second decoder extracts, for each point, an offset to the center of the instance. It also predicts an error estimate for the offsets, used later in the clustering (Section 8.1.2.6). Both decoders use a novel range-image-based trilinear upsampling, which we explain in detail in Section 8.1.2.3. Finally, the instance extractor uses the center offsets and the semantic embeddings to assign an instance id to each point in the *thing* classes and categories, before unprojecting the points to 3D. Figure 8.3 shows an example of the input range image, the semantic embedding $\widehat{\mathbf{e}}_p$ as a random projection from 32 dimensions to 3 displayed as RGB, the center offsets $\widehat{\mathbf{o}}_p$ showing each one of the offsets in x, y, and z as RGB colors, and the final outputs based on semantic and geometric embeddings.

8.1.2.1 Projection

The first step in the projective panoptic segmentation pipeline is to project the points using a spherical projection (Figure 8.3, top). To this end, we transform all (x, y, z) 3D points into a set of (u, v) 2D image coordinates using Equation (7.1).

Analogously to Chapter 7, this generates a $(5, H, W)$ volume which represents the point cloud as an image with channels $(\text{range}, x, y, z, \text{remission})$. For a point cloud of size N, we generate an index matrix of size $(N, 2)$ containing all the (u, v) image coordinate pairs, which we use later when transferring back the predictions to 3D, as explained in Section 8.1.2.7. Since the backbone is extracted from the RangeNet++ model in Chapter 7, the projection follows the same procedure.

8.1.2.2 Backbone and Point Cloud Pyramid

Relevant works for projection-based LiDAR semantic segmentation are SqueezeSeg [184, 185] and RangeNet++ [114], the latter being the first approach of this type used to tackle SemanticKITTI. These approaches exploit the way the sensor acquires the points using a rotating array of laser beams and use a 2D segmentation CNN on a spherical projection of the input point cloud.

Both of these approaches downsample periodically on the width dimension by using strided convolutions. This generates feature volumes of OS 1, 2, 4, 8, 16, and 32, which are later skipped to the decoder to recover high-frequency signals lost in downsampling, aiding the upsampling process, which uses either bilinear upsampling or transposed convolutions. However, neither of these approaches exploit the fact that the inputs contain useful metric information as they downsample, to aid the upsampling. In this work, instead, each time we apply strided convolutions, we store a downsampled version of the point cloud which contains the points at the centers of the locations where the kernels were applied. This can

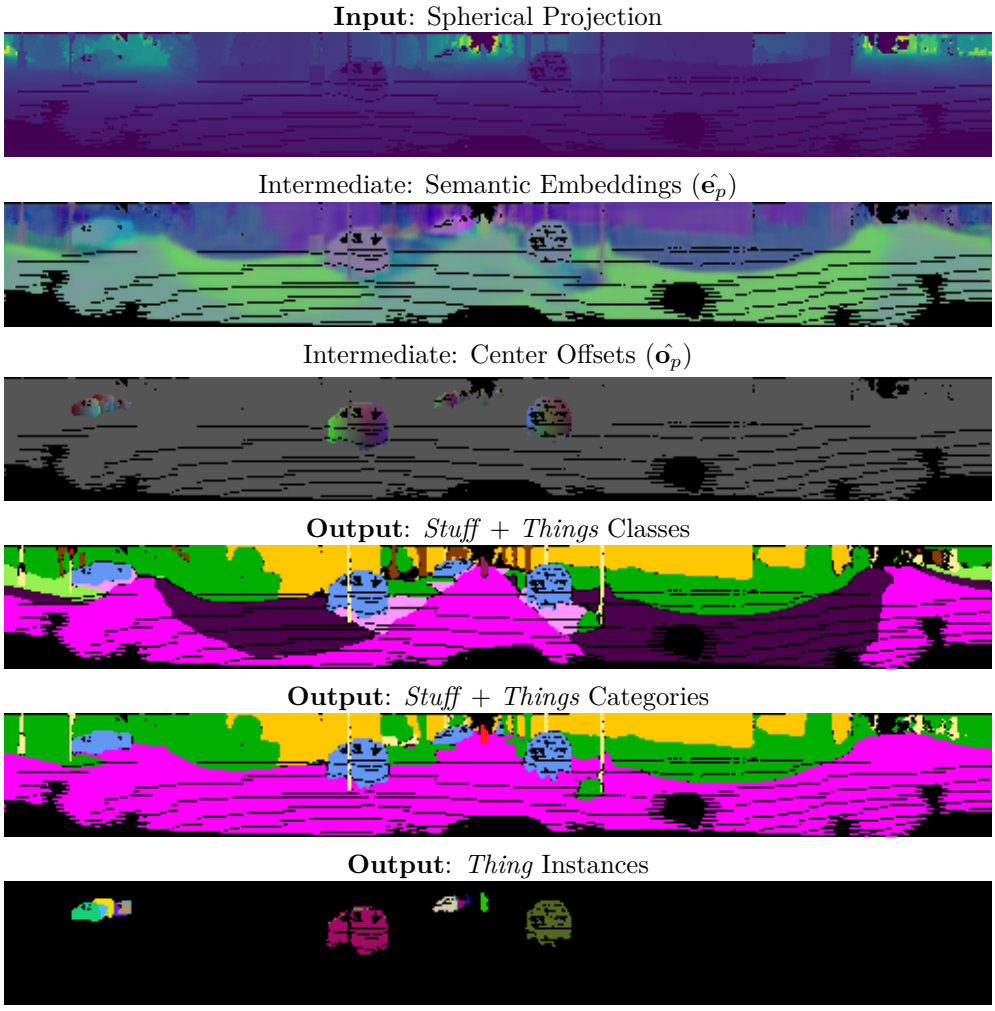

Figure 8.3: Example input, intermediate results, and outputs of our panoptic segmentation. Best viewed in color.

be used to recover useful geometric information during the upsampling in the decoder that allows us to improve the accuracy of the final output. The foundation of our architecture and shared feature extractor for both the instance head and the semantic head branch is DarkNet53 [114, 144], as in Chapter 7. The reason to pick this architecture for the backbone is two-fold. First, it is well established that the DarkNet53 architecture is very efficient for GPU deployment [144]. Second, it allows us to do a direct comparison to the semantic segmentation-only method trained in Chapter 7, which did not have the extra burden of extracting instances.

8.1.2.3 Decoders

After the point cloud range image is processed by the backbone and the sampled pyramid is generated, we pass the features through two different decoders that complement each other to solve the task of panoptic segmentation. One decoder predicts the instance centers, and the other one the semantics of the scene. Both decoders follow the same structure, illustrated in Figure 8.2.

After upsampling the feature volume, we concatenate the upsampled features with the matching resolution from the backbone as a skip connection followed by a convolutional block. These blocks also learn the task-dependent weights that separate the tasks. This is done recursively until the input resolution is met, and the task-dependent heads are applied.

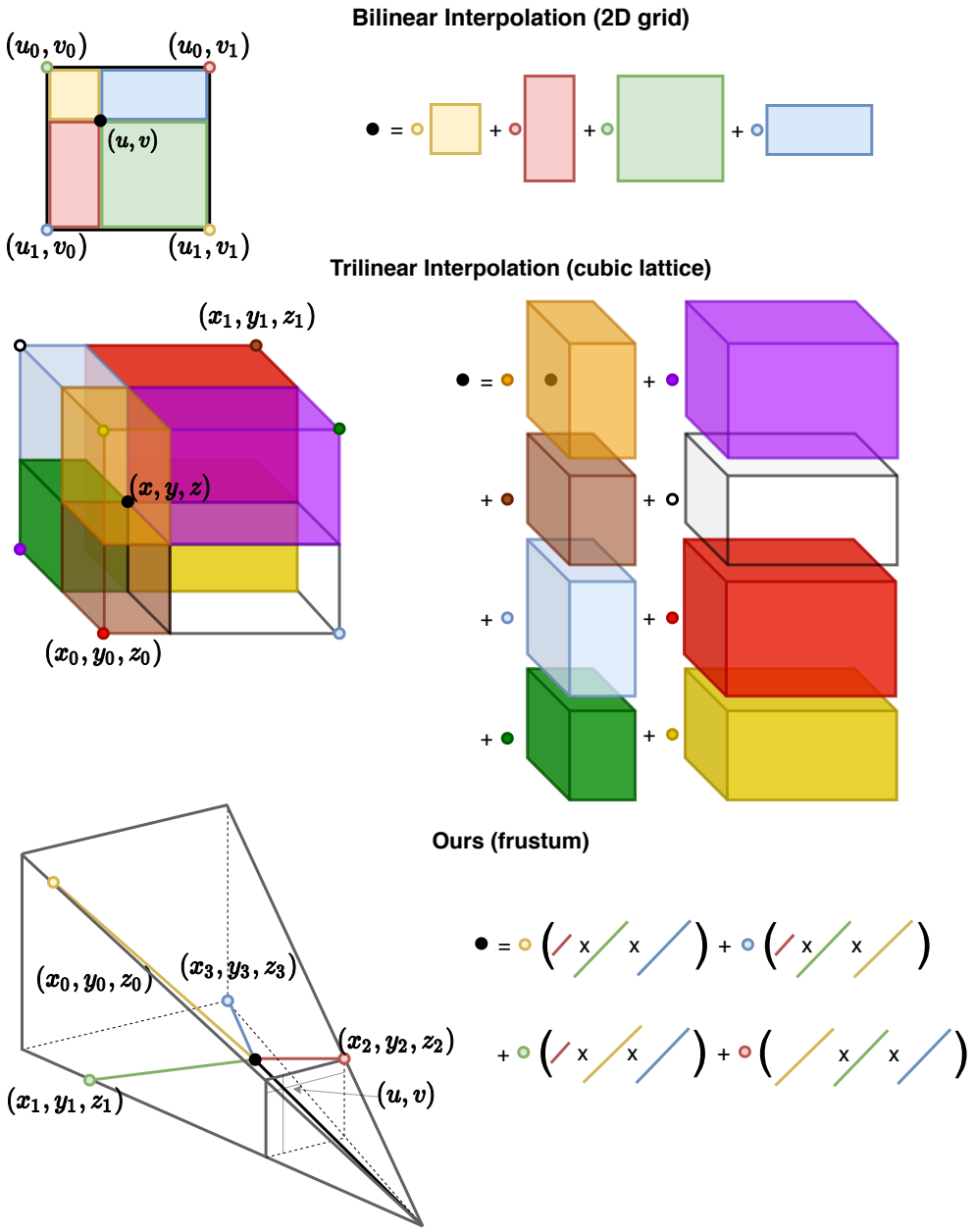

Figure 8.4: Upsampling methods graphically. The black dot corresponds to the desired interpolated feature value, while the colored dots represent the values to interpolate from. Top: Bilinear upsampling. Center: Trilinear Interpolation. Bottom: Ours. The vertex values are obtained in the range image domain, but their distances to the query point are calculated in 3D as their real Euclidean distance. Then the interpolated feature value is a linear combination of four the closest feature values in the lower resolution grid (which define the frustum), with the coefficients calculated as an inverse of the distance to the point, normalized by the total distance.

In contrast to prior works [114, 184, 185], which upsample the backbone features using transposed convolutions or bilinear upsampling, exploiting closeness in the image, we implement a differentiable trilinear upsampling layer, which exploits the fact that our inputs are 3D point clouds and not camera images. As in bilinear upsampling, for each point in the resolution that is currently upsampled, we find the 4 corresponding points in the coarser grid using the point cloud pyramid. However, since our image is actually another representation in memory of a point cloud, the real information of vicinity between the points and its coarser corresponding points is known to us through their real (x, y, z) coordinates, besides their image (u, v) coordinate. This allows us to approximate a trilinear upsampling by using the real 3D Euclidean distances rather than the 2D image ones. Even though this is technically not a strict trilinear upsampling due to the absence of a cubic lattice, our approach uses the real 3D distances. Thus, it more closely resembles the trilinear upsampling than bilinear interpolation, as shown in Figure 8.4. We now proceed to explain how our interpolation compares with the two standard ones presented.

Bilinear interpolation. Bilinear interpolation uses the information of vicinity in image coordinates for points lying in a 2D grid to interpolate the value of feature f at an off-grid location. This follows the formula

$$f(u,v) = \frac{\begin{bmatrix} v_1 - v \\ v - v_0 \end{bmatrix}^\top \begin{bmatrix} f(u_0,v_0) & f(u_1,v_1) \\ f(u_0,v_1) & f(u_1,v_0) \end{bmatrix} \begin{bmatrix} u_1 - u \\ u - u_0 \end{bmatrix}}{(u_1 - u_0)(v_1 - v_0)}, \tag{8.1}$$

where (u, v) represents the 2D coordinates of the point for which we want to interpolate the feature value, and u_0, v_0 are the limit coordinates of the anchor points we interpolate from. This is very useful (and fast) for decoders in image segmentation, since image vicinity is the only information available. In contrast, our images are 2D representations of known 3D points. Therefore, it is possible to know the real vicinity for each off-grid point to interpolate. As shown in the bottom plot of Figure 8.4, a query point which is closest to the top left corner in (u, v) does not necessarily mean real 3D vicinity, with the point being closer to the one represented in the top right corner instead. As an intuition, Figure 8.4 shows how the bilinear upsampling works geometrically, with the interpolation being a linear combination of the 4 closest grid points, and the coefficients corresponding to the normalized area of the opposite rectangle. This makes the interpolated feature value closest to the value at the closest 2D (u_x, v_x) coordinate.

Trilinear interpolation. In contrast, this type of interpolation uses the real 3D similarities to each of the discrete feature locations in a cubic lattice. This

follow the formula

$$
\begin{aligned}
&f(x,y,z) = \\
&= \frac{(z_1-z)\begin{bmatrix} y_1-y \\ y-y_0 \end{bmatrix}^\top \begin{bmatrix} f(x_0,y_0,z_0) & f(x_1,y_0,z_0) \\ f(x_0,y_1,z_0) & f(x_1,y_1,z_0) \end{bmatrix} \begin{bmatrix} x_1-x \\ x-x_0 \end{bmatrix}}{(x_1-x_0)(y_1-y_0)(z_1-z_0)} + \\
&+ \frac{(z-z_0)\begin{bmatrix} y_1-y \\ y-y_0 \end{bmatrix}^\top \begin{bmatrix} f(x_0,y_0,z_1) & f(x_1,y_0,z_1) \\ f(x_0,y_1,z_1) & f(x_1,y_1,z_1) \end{bmatrix} \begin{bmatrix} x_1-x \\ x-x_0 \end{bmatrix}}{(x_1-x_0)(y_1-y_0)(z_1-z_0)},
\end{aligned}
\tag{8.2}
$$

where (x, y, z) represents the 3D coordinates of the point for which we want to interpolate the feature value, and x_0, y_0, z_0 are the limit coordinates of the anchor points we interpolate from, in a cubic lattice. Following the same intuition as with bilinear interpolation, Figure 8.4 shows a geometrical interpretation of the trilinear upsampling method. Analogously to the 2D case, the points are interpolated as a linear combination of the function at the closest feature locations, and the coefficients represent the opposite, normalized volume to the grid point. Although this is more accurate to interpolate point cloud information like the one we use, we don't have a function that operates in a perfect cubic lattice, so instead we introduce a method that emulates this behavior operating in each range image frustum instead.

Proposed interpolation. Our interpolation works by generating a frustum for each point, generated by its four neighbors in the image, similarly to the 2D case. However, after the four points are selected, we calculate the four real Euclidean distances between the points and the feature locations in the coarser grid. These querry points come from the image pyramid built. Then the interpolation follows the formula

$$
f(\mathbf{p}) = \frac{\sum_{i=0}^{3} \left(f(\mathbf{p_i}) \prod_{j=0, i\neq j}^{3} \|\mathbf{p} - \mathbf{p_j}\| \right)}{\sum_{i=0}^{3} \left(\prod_{j=0, i\neq j}^{3} \|\mathbf{p} - \mathbf{p_j}\| \right)}
\tag{8.3}
$$

where $\mathbf{p} = (x, y, z)$ represents the 3D coordinates of the point for which we want to interpolate the feature value, and $\mathbf{p_0}$, $\mathbf{p_1}$, $\mathbf{p_2}$, $\mathbf{p_3}$ are the coordinates of the anchor points we interpolate from. The latter define the frustum used, starting from the sensor.

Figure 8.4 shows a geometric representation of this upsampling as well, where we see that the interpolated value is also a linear combination of the grid features, and the coefficients are calculated by the normalized product of all the opposite Euclidean distances. Intuitively, this product of segments represents the diagonal of the opposite volume. It is important to notice that it properly captures the vicinity in 3D of the interpolated points, as shown in the example in Figure 8.4, where bilinear upsampling would have failed.

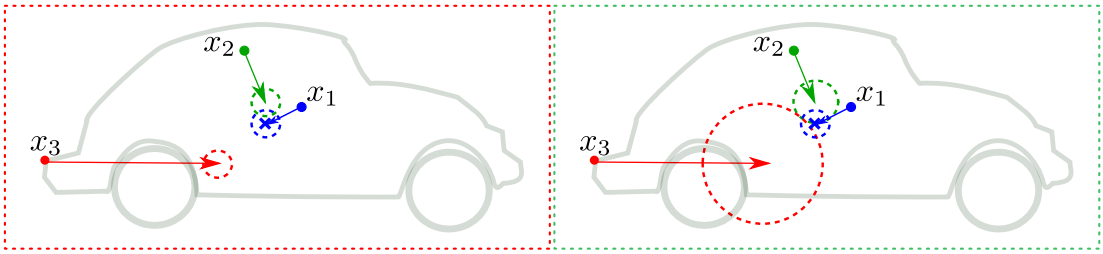

Figure 8.5: Influence of clustering radius. Left: Wrong clustering due to fixed radii. Right: Correct clustering using the learned radii depending on point difficulty.

8.1.2.4 Instance Decoder

This decoder has the responsibility to predict a 3D offset $\hat{\mathbf{o}}_p = [\Delta x, \Delta y, \Delta z]^T$ for each point $\mathbf{x}_p = [x_p, y_p, z_p]^T$ on the range image belonging to one of the *things*, from its center $\mathbf{c} = [x_c, y_c, z_c]^T$. This is similar to Neven *et al.* [120], which predicts a 2D offset to the center for instance segmentation, and Qi *et al.* [140], which predicts 3D offsets for point cloud object detection. Since we are using range image representations of 3D point clouds, our approach sits in the middle, predicting 3D offsets given an image representation. After the offsets are predicted, each point in the *thing* mask (from the semantic segmentation) predicts a center coordinate, which is used to cluster the instances. This is done using the region-based clustering method described in Section 8.1.2.6.

The offsets $\widehat{\mathbf{o}}_p$ are learnt through an L_2 loss of the form:

$$\mathcal{L}_{CENTER} = \frac{1}{I} \sum_{i=1}^{I} \frac{1}{P_i} \sum_{p=1}^{P_i} [\widehat{\mathbf{o}}_p - (\mathbf{x}_p - \mathbf{c}_i)]^2, \tag{8.4}$$

where I is the number of instances in the batch, and P_i is the number of points in the instance i. During inference, the predicted center for each point can then be calculated as $\hat{\mathbf{c}}_p = \hat{\mathbf{x}}_p - \hat{\mathbf{o}}_p$.

To use a region-based clustering method, an intra-cluster radius needs to be defined. For the center embedding $\hat{\mathbf{c}}_p$, the radius is the maximum Euclidean distance that we allow two points in the same instance to predict. In theory, this radius could be fixed and chosen by cross-validation, but research shows that different points have different levels of accuracy [120], resulting in either over- or under-segmentation. Using a learned, adaptive radius for each point instead can help solve this problem, as shown in Figure 8.5. Therefore, we add three convolutional blocks on top of the offset encoder that predict this radius $\hat{\epsilon}_p$ for each point, estimating the radius as the Euclidean distance between the offset prediction and its ground truth, for each point, during the training, i.e,

$$\mathcal{L}_{CENT\ ERR} = \frac{1}{I} \sum_{i=1}^{I} \frac{1}{P_i} \sum_{p=1}^{P_i} \left[\hat{\epsilon}_p - \|\widehat{\mathbf{o}}_p - (\mathbf{x}_p - \mathbf{c}_i)\| \right]^2. \tag{8.5}$$

The intuition behind using the training offset error as the ground truth for this variable radii is not to estimate the uncertainty of each prediction, which would not be possible since the training error is usually significantly lower than the test time error. Instead, we are interested in learning the *relative difficulty* of different points belonging to the same instance. Figure 8.5 shows an example where the point in the fender is significantly more inaccurate than the point in the center of the car, requiring a larger radius to be properly assigned to this car's cluster. This is fixable by simply using a larger radius for all points, but this leads to under-segmentation in many cases in our dataset, preventing us from a proper understanding of the scene. Therefore, at inference time we use this learned tolerances as relative thresholds for each point, but adjust the overall scale by a constant factor to account for the difference between training and validation error. This factor is the ratio between the latter.

8.1.2.5 Semantic Decoder

This decoder predicts a 32-dimensional semantic embedding for each point in the range image representation. From this embedding, two $[1 \times 1]$ convolutional heads are used to predict classes and categories. In order to train the network to predict both, we use a cross-entropy loss for the classes and an analogous one for the categories, of the form:

$$\mathcal{L}_{SEM} = -\sum_{c=1}^{C} w_c \, y_c \, \log\left(\hat{y}_c\right), \tag{8.6}$$

where $w_c = \frac{1}{\log\left(f_c + \epsilon\right)}$ is a class-wise weight calculated as a function of the inverse class-frequency f_c, and ϵ limits the largest possible weight for a class. In practice, the category prediction does not need its own head or loss. However, our experiments show that adding a separate head, along with a category loss, improves the category segmentation, effectively mapping semantically similar classes together in embedding space.

To aid the instance segmentation head center predictions, we also add an auxiliary loss that applies directly to the embeddings from the semantic head. Figure 8.3 shows a random projection from the 32-dimensional embeddings to 3 dimensions, plotted as RGB values for illustration purposes. It is possible to see that not only the embeddings are different from class to class, but also between different instances of the same class. This loss uses metric learning to cluster all pixel embeddings $\hat{\mathbf{e}}_p$ of the same instance close to each other, and pushing all mean embeddings of different instances away from each other. This is done

through an attraction and a repulsion loss, of the form:

$$L_{ATTRACT} = \frac{1}{I}\sum_{i=1}^{I}\frac{1}{P_i}\sum_{p=1}^{P_i}\left(\hat{\mathbf{e}}_{i,p} - \hat{\bar{\mathbf{e}}}_i\right)^2 \tag{8.7}$$

$$L_{REPEL} = \frac{1}{I(I-1)}\sum_{\substack{i_A=1\\ i_A\neq i_B}}^{I}\sum_{i_B=1}^{I}\frac{1}{\left(\hat{\bar{\mathbf{e}}}_{i_A} - \hat{\bar{\mathbf{e}}}_{i_B}\right)^2}. \tag{8.8}$$

where $\hat{\mathbf{e}}_{i,p}$ is the embedding for pixel p of instance i, and $\hat{\bar{\mathbf{e}}}_i$ is the mean embedding of all points in instance i.

Analogously to the center offsets, we also predict an error radius for each pixel embedding that is later used for the adaptive clustering, following an analogous loss to the center offset error loss:

$$\mathcal{L}_{EMBED\ ERR} = \frac{1}{I}\sum_{i=1}^{I}\frac{1}{P_i}\sum_{p=1}^{P_i}\left[\hat{\epsilon}_p - \left\|\hat{\mathbf{e}}_{i,p} - \hat{\bar{\mathbf{e}}}_i\right\|\right]^2, \tag{8.9}$$

8.1.2.6 Instance Extractor

To obtain all individual instance IDs, we employ an iterative procedure. First, we sample a *thing* point from the semantic prediction and obtain its distance in feature space to all other points in the same class. We propose two alternative features to do this through the instance and the semantic decoders. Using the instance decoder, the features represent the prediction of the center of the instance $\widehat{\mathbf{c}}_i = \mathbf{x}_p - \widehat{\mathbf{o}}_p$. Using the semantic decoders, the features used are the embeddings of each point $\hat{\mathbf{e}}_{i,p}$. In both cases, we then use the predictions of the error for each pixel as the clustering radius to consider points as belonging to the same instance as the initially sampled point. We then assign the instance ID to all points within the radius, remove these points from the pool, and start over with a new sampled point, until all points are consumed. In our ablation study, shown in Section 8.2.2, we compare all features and the usage of the learned radius vs. a statically defined threshold chosen by cross-validation.

8.1.2.7 Point Cloud Extraction and Post-Processing

After the segmentation of the point cloud as a range image, recovering all labels for the N original points in the point cloud is desired. However, if $N > HW$, unprojecting the image using the intrinsic calibration of the sensor does not reconstruct all points. This is why the reprojection of the labeled points to the 3D world is, instead, performed by keeping an $(N, 2)$ shaped list of (u, v) image indexes that can use the label image as a lookup table to recover the labels and IDs for all N points.

8.2 Experimental Evaluation

In the first part of the experiments, we compare our panoptic segmentation approach with state-of-the-art approaches on our SemanticKITTI dataset, a large-scale LiDAR dataset providing instance and semantic segmentation annotations described in Chapter 6, as well as a panoptic segmentation benchmark. We then provide ablations studies to show the importance of different design decisions of our approach, such as the use of semantic and center embeddings jointly, the inclusion of the novel trilinear upsampling module, and the category loss.

Implementation details. In all the following experiments, we use the following parameters, when not otherwise stated. All networks in the single-stage approach were trained following the same training schedule, using Adam optimizer with a learning-rate of 0.001, a warm-up ramp of 1 epoch, momentums $(0.9, 0.99)$, a learning-rate decay of 0.99 per epoch, training for 200 epochs. To integrate all losses, we tested GradNorm [28], but yielded no significant improvement over the simple addition of all losses.

Dataset. We evaluate our approach on our SemanticKITTI dataset, which provides point-wise semantic as well as temporally consistent instance annotations for all scans of the KITTI odometry split [46]. The dataset provides 23 201 scans for training, and the remaining 20 351 scans are used for evaluation on a benchmark server. We use sequence 08 from the training data comprised of 4 071 scans for validation purposes. The dataset contains overall 28 classes from which the vehicle classes and classes representing humans have point-wise instance annotations.

Evaluation Metrics. In order to compare the semantic segmentation branch with other approaches in the semantic segmentation benchmark, we calculate the mean intersection over union (mIoU) over all classes, defined in Equation (6.1), and for the panoptic task we use the metrics defined in the benchmark, in Equation (6.3), and Equation (6.4).

SemanticKITTI also defines an ontology assigning each class to a category, e.g., *truck*, *car*, *other-vehicle* belong to the category *vehicle* [11]. These category definitions are useful for autonomous driving, e.g., identifying *humans* as an alternative to the more fine-grained *person* or *bicyclist* classes. Therefore, alternatively to the class-wise metrics, we also evaluate all approaches with respect to categories.

8.2.1 Comparison to the State of the Art

The first experiment evaluates the performance of our single-stage approach in comparison with the two-stage approaches proposed as baselines for the panoptic task defined in Section 6.3.3.

Method	FPS	mIoU	PQ	PQ†	RQ	SQ	PQ^{Th}	RQ^{Th}	SQ^{Th}	PQ^{St}	RQ^{St}	SQ^{St}
KPConv [174] + PointPillars [82]	1.9	58.8	44.5	52.5	54.4	80.0	32.7	38.7	81.5	53.1	65.9	79.0
RangeNet++ [114] + PointPillars [82]	2.4	52.4	37.1	45.9	47.0	75.9	20.2	25.2	75.2	49.3	62.8	76.5
Ours (without category loss)	11.8	51.0	35.3	44.3	45.0	76.5	19.1	24.1	76.7	47.2	60.2	76.4
Ours (with category loss)	11.8	50.9	38.0	47.0	48.2	76.5	25.6	31.8	76.8	47.1	60.1	76.2

Table 8.1: Comparison of test set results on SemanticKITTI using *stuff*(St) and *thing*(Th) *classes*. All results in [%].

Method	FPS	mIoU	PQ	PQ†	RQ	SQ	PQ^{Th}	RQ^{Th}	SQ^{Th}	PQ^{St}	RQ^{St}	SQ^{St}
KPConv [174] + PointPillars [82]	1.9	84.8	70.0	71.5	79.2	87.3	54.3	62.1	86.7	77.8	87.8	87.7
RangeNet++ [114] + PointPillars [82]	2.4	78.8	63.6	65.9	74.3	83.8	43.8	52.3	82.2	73.5	85.3	84.6
Ours (without category loss)	11.8	77.4	59.9	62.6	71.4	81.7	37.7	47.2	78.0	71.0	83.4	83.5
Ours (with category loss)	11.8	77.8	65.8	68.1	77.2	83.5	53.6	63.8	82.5	71.9	84.0	84.0

Table 8.2: Comparison of test set results on SemanticKITTI using *stuff*(St) and *thing*(Th) *categories*. All results in [%].

Table 8.1 shows the results in terms of class-wise performance on the test set. Likewise, Table 8.2 shows the performance in respect to the categories. Our proposed single-stage approach gets superior performance in comparison with the two-stage approach using RangeNet++ [114]. However, it is worse than the best performing KPConv [174], which achieves 44.5 panoptic quality and can be mainly attributed to better semantic segmentation, albeit at a higher computational cost.

We also show in the results the difference between using the category loss and head, vs. a lookup table between class prediction and corresponding category. Interestingly, these results show that the category loss helps to improve the panoptic quality performance for *thing* classes, but leads to worse results for the semantic segmentation quality as shown by a drop in mIoU.

The main motivation for our single-stage approach is the improved computational efficiency in comparison to the aforementioned two-stage approaches. Instead of using multiple different networks, we can use a single multi-task network, which also profits from sharing the encoder between different tasks. Figure 8.6 shows the runtime performance in relation to the panoptic quality. Clearly, our single-stage approach is considerably faster than the two-stage approaches. Here, we assume that the separate object detectors runs in parallel (314 ms for *pedestrian/cyclist* and 105 ms for *car*) after the semantic segmentation (200 ms for KPConv and 95 ms for RangeNet++) resulting in 514 ms and 409 ms respectively. Our single-stage approach with trilinear upsampling takes 85 ms on average.

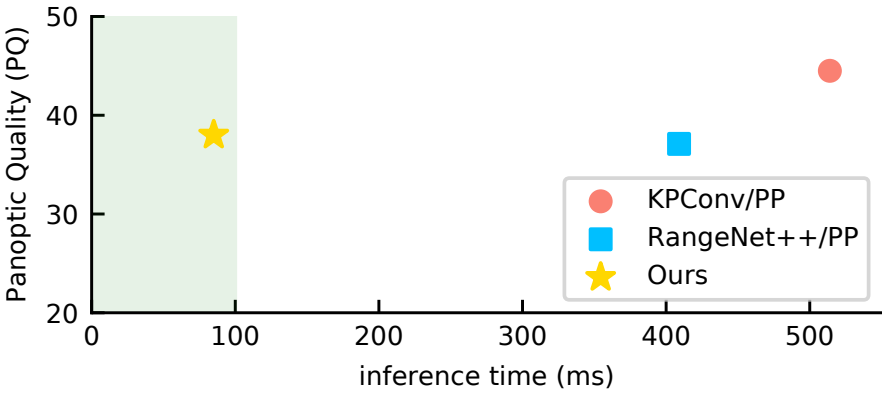

Figure 8.6: Runtime of the evaluated approaches. Green area represents the zone of approaches faster than the rate of the sensor.

OS	Method	mIoU	PQ	PQ†	FPS
32	Transpose convolution	46.4	30.8	41.9	11.1
8	Transpose convolution	48.5	31.2	42.6	4.2
32	Ours (trilinear)	50.7	36.5	46.1	11.8

Table 8.3: Influence of upsampling method evaluated on validation set with respect to panoptic quality and runtime.

8.2.2 Ablation Studies

We also validate that our contributions lead to an increase in performance with respect to panoptic quality through ablation studies. Note that here we evaluate all approaches on the validation set. The first experiment (Table 8.3), shows the influence of the upsampling method to regain spatial resolution in the decoder after the backbone downsampling. We can see that using our trilinear evaluation leads to considerable gains in class and computational performance, allowing for more downsampling without sacrificing accuracy or speed.

The next experiment (Table 8.4), shows the influence of features used for clustering instances. We can see that the best method using a single head is through the prediction of the center instances, rather than the semantic embeddings. However, combining both yields an increase in the performance of the

Center	Embedding	Learnt Radius	RQ^{Th}	PQ
✓			23.7	34.4
✓		✓	26.7	35.8
	✓		18.0	32.8
	✓	✓	20.0	33.3
✓	✓	✓	28.2	36.5

Table 8.4: Features used for clustering of *things* on validation set with respect to recognition and panoptic quality.

clustering, which is appreciated by an increase in the recognition quality of *things*. Furthermore, we compare the clustering using each feature with the learned radii vs. the best static threshold, found by cross-validation, and we show that learning a point-wise radius helps the performance of the approach.

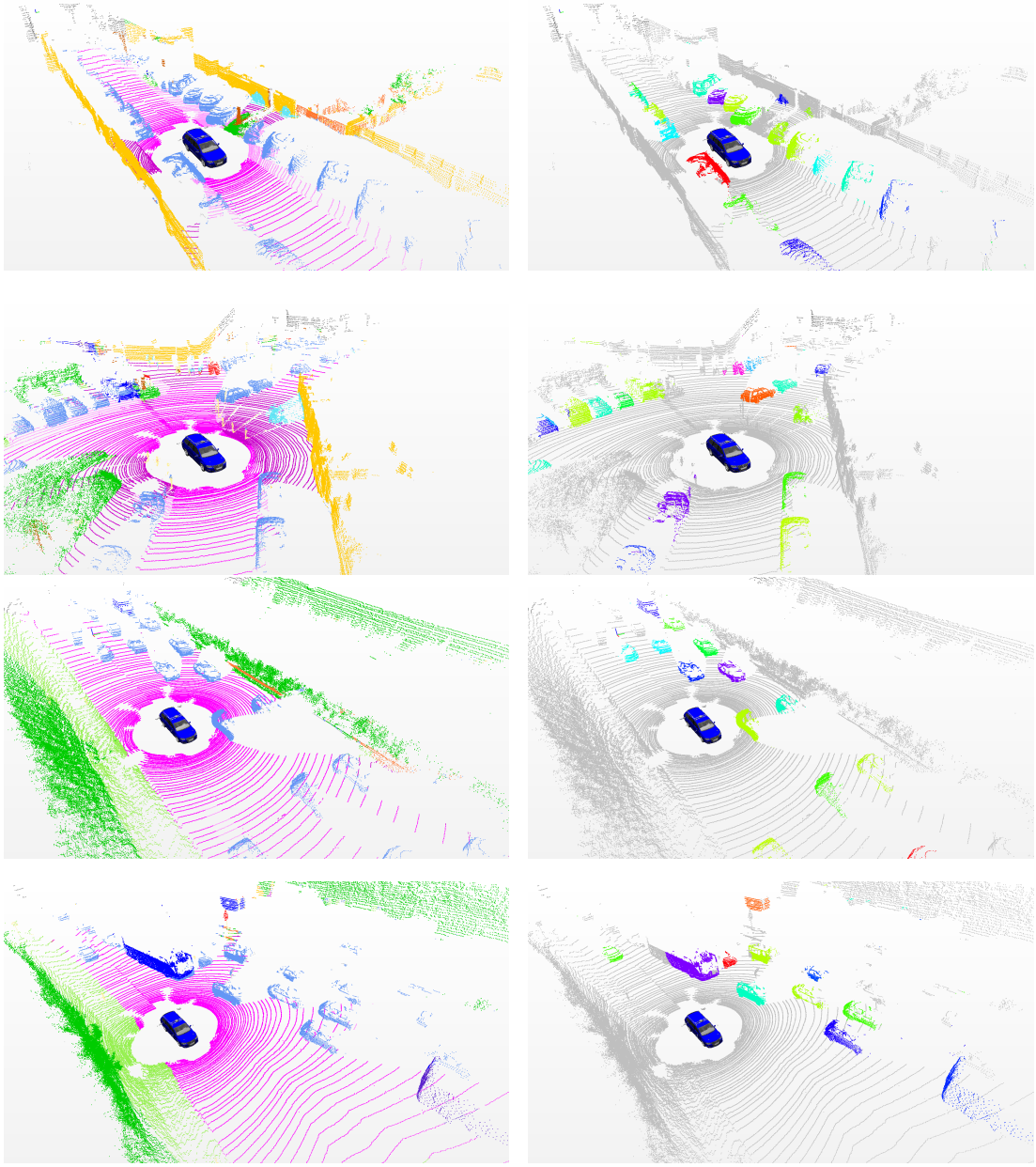

Figure 8.7: Qualitative examples of predictions by our single stage approach on sequence 13 (city driving, test set), and sequence 20 (highway driving, test set). Left: Semantics. Right: Instances.

8.3 Qualitative results

Figure 8.7 shows examples of the proposed panoptic segmentation using our single stage approach, both for the cases of urban and highway driving. These show how the results of the approach look like, for readers who are not acquainted with panoptic segmentation metric numbers. Interactive demos are available with the code, along with prediction files and trained models.

8.4 Related Work

We aim to provide a broad overview of closely related instance and semantic segmentation approaches for point clouds, but we also discuss closely related RGBD and image-based approaches.

Semantic Segmentation. For semantic segmentation of point clouds, a variety of approaches have been proposed. Voxel-based methods transform the point cloud into a voxel-grid and apply convolutional neural networks with 3D convolutions for object classification [104] and semantic segmentation [68]. Both approaches were among the first investigating such models and allow for directly exploiting architectures and insights known from image-based methods.

To overcome the limitations of the voxel-based representation, such as the inherent higher memory consumption with increasing voxel grid resolution, approaches either upsample voxel-predictions [172] using a conditional random field (CRF) or use a different representation, such as more efficient spatial subdivisions [51, 77, 150, 182, 191], graphs [81, 173], splats [166], or points directly [39, 52, 67, 72, 139, 141, 149, 174]. Opposite to these spatial partition approaches, methods exploiting the organization of the generated measurements by a rotating automotive LiDAR sensor [184, 185] or approaches using a cylindrical or spherical projection [114] of the point cloud showed promising results on the KITTI Vision Benchmark and its extension SemanticKITTI [9]. Compared to the aforementioned point cloud-based approaches, these techniques can use larger backbones [144] and realize more efficient neighbor searches by exploiting the organization of the data from the sensor directly [114].

We base our single-stage approach on the latter style, and we present a novel range-image-based tri-linear upsampling method in our decoders that exploits the image representation of neighbor information but uses the actual distances between points from the point cloud pyramid to upsample features spatially. Furthermore, we exploit a category loss that exploits the knowledge of a useful dataset ontology to improve the accuracy of the results.

Instance Segmentation. For image-based instance segmentation, there are mainly two types of approaches: detection-based [59, 63, 188] and clustering-

based [29, 112, 120, 16]. Detection-based approaches, pioneered by Mask R-CNN [59], first locate objects using an object detector and then segment the object inside the bounding box. Clustering-based approaches, pioneered by Brabandere *et al.* [16], use metric learning to find an embedding, which facilitates the clustering of pixels from an instance. Often, this also involves the prediction of a seed pixel or point, like the center of an object [29, 112, 120], from which the clustering is seeded.

Also, point cloud instance segmentation has been explored. The approach of Wang *et al.* [179] extracts point-wise features using a PointNet, which are used to generate a similarity matrix, a confidence map, and a semantic segmentation, then used to cluster instances by the similarity scores. The two-stage approach of Hou *et al.* [63] regresses bounding boxes for objects and uses then information from point clouds, but also image information to generate an instance mask for each bounding box. In contrast, the single-stage approach of Yang *et al.* [188] directly estimates a fixed number of bounding boxes and associates each bounding box with a point-wise mask separating the object from the background. Yi *et al.* [189] use object-like proposals instead of bounding boxes, which are then used to generate bounding boxes, segmentation masks, and classification into object classes.

Panoptic Segmentation. Recently, the task of panoptic segmentation [76], i.e., jointly predicting a semantic segmentation of *stuff* classes and instance segmentations for *things* gained significant interest using images [29] or RGBD data [63, 132, 180]. Panoptic segmentation metrics were also adopted by several of the major image datasets [30, 91, 119]. Most recently, even the images of KITTI were given the panoptic treatment in [115], providing a panoptic dataset and benchmark that is analogous to our SemanticKITTI, albeit for RGB images.

The approach of Pham *et al.* [132] uses a PointNet-based network to provide semantic class probabilities, but also instance embeddings. These are then used by a conditional random field [80] to predict instance labels and semantic labels for an RGBD scan. Similarly, Wang *et al.* [180] use an encoder with two decoders to generate semantic and instance features using PointNets for RGBD data, combining these with an associative segmentation module that uses semantics to generate instance IDs and vice versa. Finally, Hurtado *et al.* [69], extend the panoptic task in SemanticKITTI in combination with multi-object tracking, adding the time dimension to the problem to generate the new task of multi-object panoptic tracking (MOPT).

In contrast to prior work, we propose a single-stage, end-to-end trainable, and real-time capable approach using point clouds generated by a rotating automotive LiDAR. Our approach combines a suite of practices that improve panoptic quality. This includes the combination of semantics and geometry for instance clustering,

a learned point-wise threshold for said clustering, a new trilinear upsampling for range images in the decoders, and a joint class plus category loss.

8.5 Conclusion

In this work, we propose a novel approach for single-stage LiDAR-based panoptic segmentation which achieves high panoptic quality while still running over the frame rate of the sensor. Our approach uses a shared encoder and two decoders to infer the semantics of the environment and the offsets of each point to the center of the object it belongs to. By combining the semantic predictions with the inferred centers from the instance offset head we obtain semantic labels for all "stuff" classes, as well as instance IDs for all "thing" instances. Our experiments show that our approach achieves results that are on par with, but faster than the best performing real-time approach on SemanticKITTI. Our ablations studies also show that the addition of the novel range-image-based trilinear upsampling module allows our approach to using larger output strides than approaches using transpose convolutions, resulting in faster runtime without sacrificing accuracy. Furthermore, we show that the combination of our geometric and semantic feature embeddings helps increase the performance of the approach in terms of recognition quality. This is also the case for our learned point-wise radii, which adapts the clustering threshold for points of different difficulty. Finally, the addition of a category loss makes category-based results more robust. This means that if we get a class wrong, we will more likely confuse it with an instance of the same category, which is desired behavior. The code for this approach is available as open-source software at `http://www.github.com/PRBonn/lidar-bonnetal`. This approach, along with its availability as open-source software, provides the self-driving software stack with accurate semantic scene understanding of the finest level of detail, with both semantics and instances considered. Furthermore, because it also achieves this quickly and efficiently, the approach can run on hardware mounted on the car, enabling a wide variety of downstream tasks, from localization and mapping to obstacle avoidance, among others mentioned all throughout this thesis.

Part III

Effective and Efficient Deep Learning Software for Robotics

Chapter 9

A Robotics Software Suite for Semantic Understanding

Perception is an essential building block for most robots. In Part I and Part II of this thesis, we argue that autonomous systems need the capability to analyze their surroundings in order to safely and efficiently interact with the world, and we provide a suite of methods to achieve this. Furthermore, we stated that augmenting the robot's camera data (as well as other modalities such as LiDAR) with the semantic categories of the objects present in the scene has the potential to aid localization [4, 6, 135], mapping [75, 168], path planning and navigation [36, 194], manipulation [15, 161], precision farming [98, 111, 110] as well as many other tasks and robotic applications. Semantic segmentation is one of the topics which we covered thoroughly throughout this thesis since it provides a pixel-accurate category mask for a camera image or an image stream. The fact that each pixel in the images is mapped to a semantic class allows the robot to obtain a detailed semantic view of the world around it and aids to understand the scene.

Most methods which represent the current state of the art in semantic segmentation use fully convolutional neural networks, as we have presented in this thesis. The success of neural networks for many tasks from machine vision to natural language processing has triggered the availability of many high-quality open-source development and training frameworks such as TensorFlow [1] and PyTorch [127]. Even though these frameworks have simplified the development of new networks and the exploitation of GPUs dramatically, it is still non-trivial for a novice to build a usable pipeline from training to deployment in a robotic platform. Even for practitioners in the field, it is not trivial to decide how to get started in order to make deep learning pipelines scale. This leads to wasted time and resources through problems such as old model weights whose architectures change and therefore can no longer be used, difficult on-robot inference, repro-

Figure 9.1: Sample predictions from Bonnet. Left: Raw RGB images. Right: Overlay with semantic segmentation label from CNN prediction. From top to bottom: Cityscapes dataset [30], person segmentation inferring a photo from our research group, trained on COCO [91], Crop-Weed agricultural dataset [22]. Best viewed in color.

ducibility impossibility, and other problems. Furthermore, companies such as NVIDIA and Intel have furthermore developed custom accelerators such as TensorRT or the Neural Compute SDK. Both use graphs created with TensorFlow, Caffe, or PyTorch as inputs and transform them into a format in which inference can be accelerated by custom inference hardware. As with the other frameworks, their learning curve can be steep for a developer that actually aims at solving a robotics problem but which relies on the semantic understanding of the environment. Last but not least, source code from computer vision research related to semantic segmentation is often made available, which is a great achievement. Each research group, however, uses a different framework, and as we already mentioned, adapting the trained networks to an own robotics codebase can sometimes take a considerable amount of development time.

Therefore, we saw the need for a tool that allows a developer to easily train and deploy semantic segmentation networks for robotics. Such a tool should allow developers to easily add new research approaches into the robotic system

while avoiding the effort of re-implementing them from scratch or modifying the available code until it becomes at least marginally usable for the research purpose. This is something that we experienced ourselves and observed in the community too often.

Therefore, in this chapter, we shift the focus away from application-oriented improvements through algorithmic design. Instead, we focus on the design of a software architecture that makes deep learning easily accessible to roboticists. The main contribution showcased in this chapter is a stable, easy to use, software tool with a modular codebase, which implements semantic segmentation using CNNs. It solves training and deployment on a robot. Thus, we do not propose a new CNN algorithm or architecture here. Instead, we provide a clean and extensible implementation to make this technology easily usable in robotics and to enable a larger number of people to use CNNs for semantic segmentation on their robots. Our tool allows the scientific robotics community to save time on the CNN implementations, enabling researchers to spend more time to focus on how such information can aid robot perception, localization, mapping, path planning, obstacle avoidance, manipulation, safe navigation, etc.

We show this with different example use cases from the community, where robotics researchers with no expertise in deep learning were able to, using Bonnet, train and deploy semantics in their systems with minimal effort. Bonnet relies on TensorFlow for our graph definition and training but provides the possibility of using different backends with a clean and stable C++ API for deployment. It allows for the possibility to transparently exploit custom hardware accelerators that become commercially available, without modifying the robotics codebase.

Although we do not propose a new scientific method, we believe that this work has had a strong positive impact on the robotics community. Bonnet has a considerable user base and won "Best Demo Award" at the Workshop on Multimodal Robot Perception at ICRA 2018. Our open-source software is available at `https://github.com/PRBonn/bonnet`.

9.1 Bonnet

In this section, we present our semantic segmentation tool called Bonnet with a Python training pipeline and a C++ deployment library. The C++ deployment library can be used standalone or as a ROS node. We provide three sample architectures focusing on realtime inference, based of ERFNet [151], see Figure 9.2, InceptionV3 [170], and MobilenetsV2 [159] as well as pre-trained weights on four different datasets. Our codebase allows for fast multi-GPU training, for easy addition of new state-of-the-art architectures and available datasets, for easy training, retraining, and deployment in a robotic system. It furthermore allows for

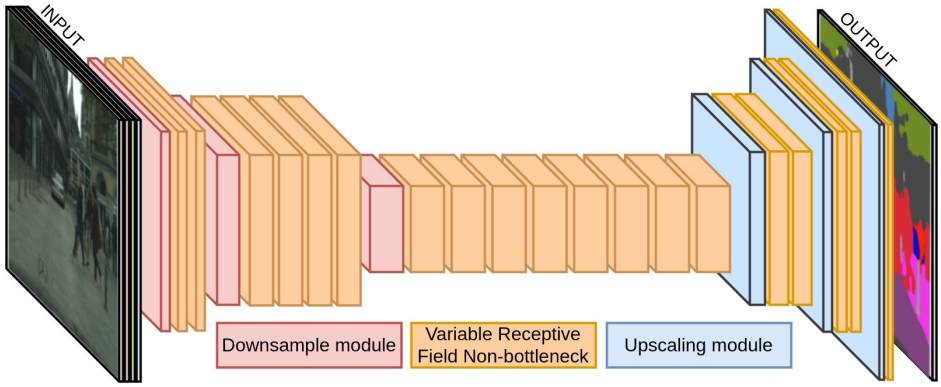

Figure 9.2: Example of an encoder-decoder semantic segmentation CNN implemented in Bonnet. It is based on the non-bottleneck idea behind ERFNet [151]. Best viewed in color.

transparently using different backends for hardware accelerators as they become available. This all comes with a stable C++ API.

The usage of Bonnet is split into two steps. First, training the models to infer the pixel-accurate semantic classes from a specific dataset through a Python interface which is able to access the full-fledged API provided by TensorFlow for neural network training. Second, deploying the model in an actual robotic platform through a C++ interface which allows the user to infer from the trained model in either an existing C++ application or a robot using the ROS operating system. Figure 9.3 shows a modular description of this division, from the application level to the hardware level, which we explain in detail in the following sections. Note that for a reasonable number of use-cases, a developer using Bonnet can avoid coding more or less completely. By simply providing own training data, a new application can be deployed in a robot by simply fine-tuning one of the models and deploying using the ROS node.

In sum, we provide (i) a modular implementation platform for training and deploying semantic segmentation CNNs in robots; (ii) three sample architectures that perform well for a variety of perception problems in robotics, while working roughly at sensor framerate; (iii) a stable, easy to use, C++ API that also allows for the addition of new hardware accelerators as they become available; (iv) a way to promptly exploit new datasets and network architectures as they are introduced by computer vision and robotics researchers.

9.2 Bonnet Training

The training of the models is performed through the methods defined through the abstract classes `Dataset` and `Network`, see Figure 9.3. These handle the prefetching, randomization, and pre-processing of the images and labels, and the supervised training of the CNNs, respectively.

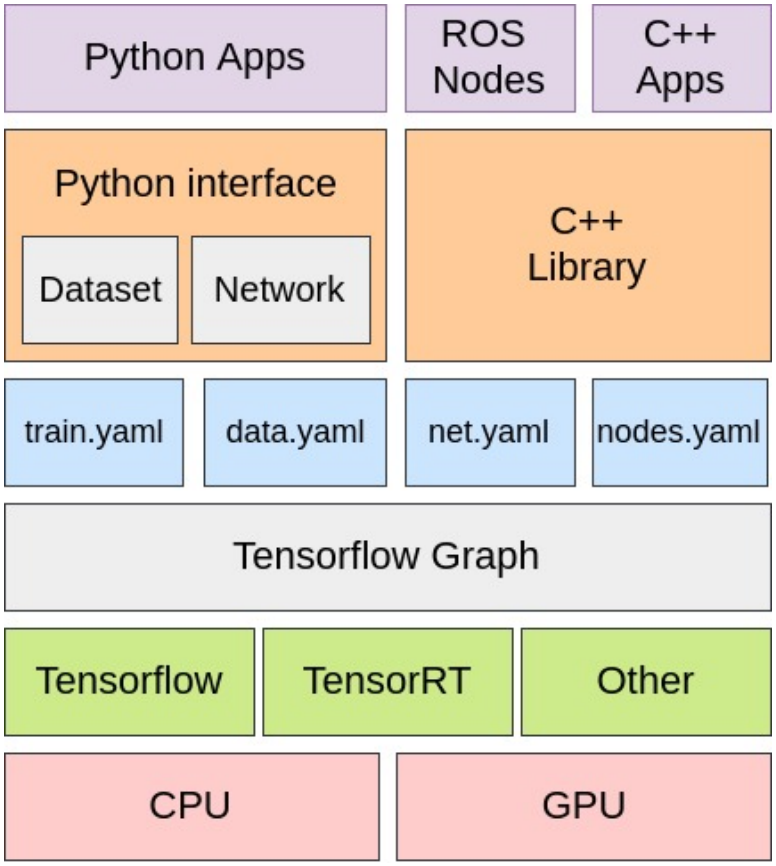

Figure 9.3: Abstraction of the codebase. Python interface is used for training and graph definition, and C++ library can use a trained graph and infer semantic segmentation in any running application, either linking it or by using the ROS node. Both interfaces communicate through the four configuration files in *yaml* format and the trained model weights.

In order to train a model using our tool, there is a sequence of well-defined steps that need to be performed, which are:

- Dataset definition, which is optional if the dataset is provided in one of our defined standard dataset formats.
- Network definition, which is also optional if the provided architecture fits the needs of the addressed semantic segmentation task.
- Hyper-parameter tuning.
- GPU training, either through single or multiple GPUs. This step can be performed either from scratch or from a provided pre-trained model.
- Graph freezing for deployment, which optimizes the models to strip them from training operations and provides an optimized model format for each supported hardware family.

9.2.1 Dataset Definition

The abstract class `Dataset` provides a standard way to access dataset files, given a desired split for it in training, validation, and testing sets. The codebase contains a general dataset parser, which can be used to import a directory containing images and labels that are split into our standard dataset format. This parser can also be used as a guideline to implement a different parser, for an own organization of the dataset files. The definition of each semantic class, the colors for the debugging masks, the desired image inference size, and the location of the dataset are meant to be performed in the corresponding dataset's *data.yaml* configuration file, of which there are several examples in the codebase. Once the dataset is parsed into the standard format, the abstract class `Network` knows how to communicate with it in order to handle the training and inference of the model. Besides the handling of the file opening and feeding to the CNN trainer, the abstract dataset handler performs the desired dataset augmentation, such as flips, rotations, shears, stretches, and gamma modifications. The dataset handler runs on a thread different from the training, such that there is always an augmented batch available in RAM for the network to use, but also allows the program to use big datasets in workstations with limited memory. The selection of this cache size allows for speed vs. memory adjustment, which depends on the system available to the trainer.

9.2.2 Network Definition

Once the dataset is properly parsed into the standard format, the CNN architecture has to be defined. We provide three sample architectures and provide pre-trained weights for different datasets and different network sizes, depending on the complexity of the problem. Other network architectures can be easily added, given the modular structure of our codebase, and it is the main purpose of the tool to allow the implementation of new architectures as they become available. For this, the user can simply create a new architecture file, which inherits the abstract `Network` class and defines the graph using our library of layers. If a novel layer needs to be added, it can be implemented using TensorFlow operations. The abstract class `Network`, see Figure 9.3, contains the definition of the training method that handles the optimization through stochastic gradient descent, inference methods to test the results, metrics for performance assessment, and the graph definition method, which each architecture overloads in order to define different models. If a new architecture requires a new metric or a different optimizer, these can be modified simply by overloading the corresponding method of the abstract class. The interface with the model architecture is done through the *net.yaml* configuration file, which includes the selection of the archi-

tecture, the number of layers, number of kernels per layer, as well as some other architecture-dependent hyper-parameters such as the amount of dropout [61], and the batch normalization [70] decay.

The interface with the optimization is done through the *train.yaml* configuration file, which contains all training hyper-parameters, such as learning rate, learning rate decay, batch size, the number of GPUs to use, and some other parameters such as the possibility to periodically save image predictions for debugging, and summaries of the weights and activations histograms, which take a lot of disk space during training, and are only useful to have during hyper-parameter selection. There are examples of these configuration files provided for the included architectures in the codebase.

It is important to notice that since the abstract classes `Network` and `Dataset` handle most cases well with their default implementation, no coding is required to add a new task and train a model unless for special cases. However, if a complex dataset is to be added or a new network implementation is desired, Bonnet allows for its easy implementation.

9.2.3 Hyper-parameter Selection

Once the network and the dataset have been properly defined, the hyper-parameters need to be tuned. We recommend doing the hyper-parameter selection through random-search, as single GPU jobs, which can be performed by starting the training with different configuration files (*net.yaml*, *train.yaml*), with all summary options enabled, and then choosing the best performing model for a final multi-GPU training until convergence. The tool is designed in this way for more simplicity, and because the hyper-parameter selection jobs can be scheduled easily with an external job-scheduling tool. Hyper-parameters that can be configured are: the number of images to cache in RAM, the amount and type of data augmentation, the decays for batch normalization [70], the regularization through weight-decay and dropout [61], the learning rate and momentums for the optimizer, the type of weighting policy for dealing with unbalanced classes in the dataset, the γ for the focal loss [92], the batch size, and number of GPUs.

9.2.4 Multi GPU training

Once the most promising model is found, the training can be done with this hyper-parameter set using multiple GPUs to be able to increase the batch size, and hence, the speed of training. Changing the number of GPUs used for training is as simple as changing the setting in the *train.yaml* configuration file, but we recommend scaling the hyper-parameter set found following the procedure described by Goyal *et al.* [50] for better results. The multi-GPU training, as

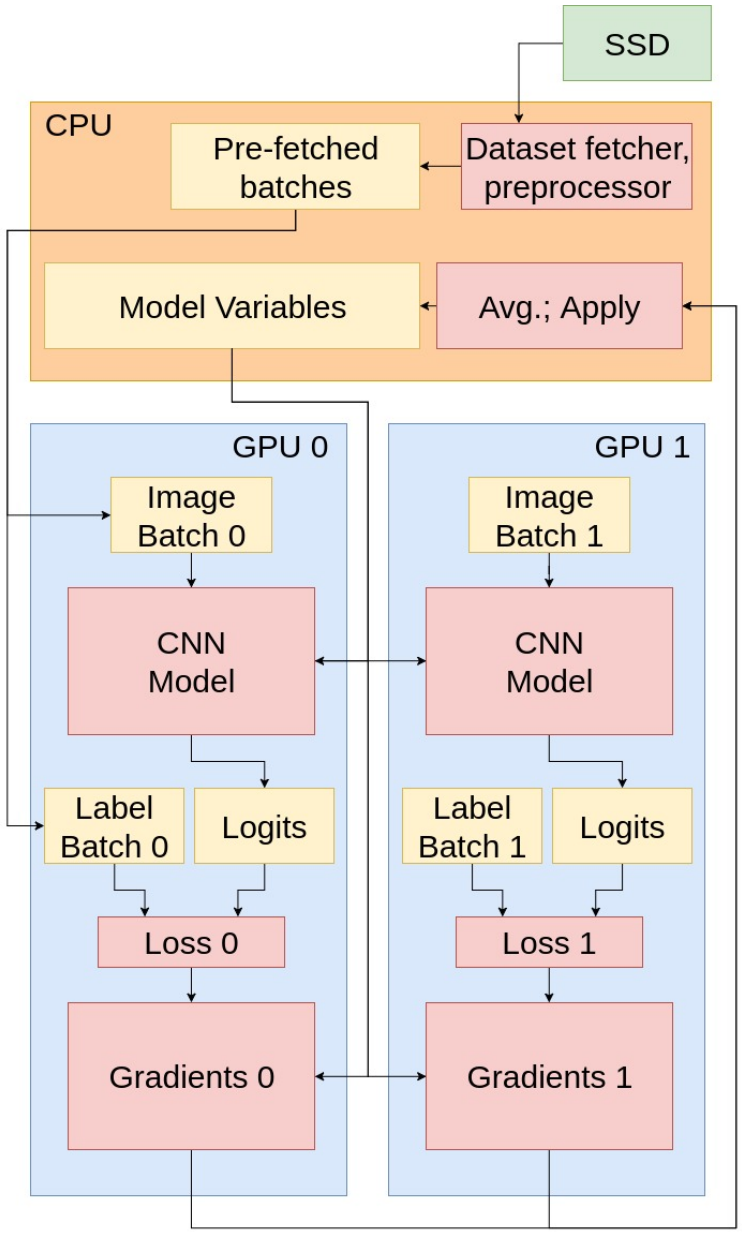

Figure 9.4: Multi-GPU training. Example using two GPUs, but scalable to all GPUs available in workstation.

described in Figure 9.4, is performed by synchronously averaging the gradients obtained by a single stochastic gradient descent step in each GPU. For this, all model parameters are stored in the main memory and they are transferred to each GPU after each step of averaged gradient update. This is handled by the abstract network's training method, and it is transparent to the user. The accuracy and mean intersection over union (IoU, see Equation (6.1)) are periodically reported and the best performing models in the validation set are stored. We store both the best accuracy and the best intersection over union model, for posterior use in deployment.

Another important work to make GPU training more efficient is the introduction of the concept of "checkpointed gradients" [25], which allows fitting big models in GPU memory in sub-linear space. This is done by checkpointing nodes in the computation graph defined by the model, and recomputing the parts of the graph in-between those nodes during backpropagation. This makes it possible to calculate the network gradients in the backward pass at reduced memory cost, without increasing the computational complexity linearly. Our tool allows us to use the implementation of the checkpointed gradients, and therefore, besides allowing for bigger batches due to the multi-GPU support, it also allows for bigger per-GPU batches.

9.2.5 Graph Freezing for Deployment

Once the trained model performs as desired, the tool exports a log directory containing a copy of all the configuration files used, for later reference, and two directories inside containing the best IoU and best accuracy checkpoints. To deploy the model and use it with different back-ends, such as TensorRT, we need to "freeze" the desired model. Freezing removes all of the helper operations required for training and unnecessary for inference, such as the optimizer ops, the gradients, dropout, and calculation of train-time batch normalization momentums. The abstract network provides a method which handles this procedure and creates another directory with four frozen models: the model in NCHW format (also called "channel-first" in the literature, referring to the memory alignment of the batch), which is faster when inferring using GPUs; the model in NHWC format (also called "channel-last"), which can be faster when using CPUs; an optimized model, which tries to further combine redundant operations, and an 8-bit quantized model for faster inference. This method also generates a new configuration file called *nodes.yaml*, which contains important node names, such as the inputs, code, and outputs as logits, softmax, and argmax. This allows for a more automated parsing of the frozen model during inference and automatically remembering the names of the inputs and outputs. We provide a Python script for this procedure, which takes a training log directory as an input and outputs all the frozen models and their configuration files in a packaged directory that contains all files needed for deployment. We also provide other applications to test this model in images and videos, in order to observe the performance qualitatively for debugging and to serve as an example for serving using python, in case this is desired. It is key to notice that since the whole process can be performed in a host PC, the device PC on the robot only needs the dependencies to run the inference, such as our C++ library.

```
#include <bonnet.hpp>
#include <opencv2/core/core.hpp>
#include <string>

int main() {
  // path to frozen dir
  std::string path = "/path/frozen/pb";
  // tf for Tensorflow, trt for TensorRT
  std::string backend = "trt";
  // gpu or cpu (or specialized)
  std::string dev = "/gpu:0";

  // Create the network
  bonnet::Bonnet net(path, backend, dev);
  // Infer image from disk
  cv::Mat image, mask, mask_color;
  image = cv::imread("/path/to/image");
  net.infer(image, mask);
  // If necessary, colorize (like Fig.1)
  net.color(mask, mask_color);
}
```

Figure 9.5: C++ code showing simplicity of semantic segmentation CNN inference in C++ application, using Bonnet tool as a library.

9.3 Bonnet Deployment

For the deployment of the model on a real robot, we provide a C++ library with an abstract handler class that takes care of the inference of semantic segmentation, and allows for each implemented back end to run without changes in the API level. The library can handle inference from a frozen model that is generated through the last step of the Python interface. Bonnet handles the inference through the user's selection of the desired back end, execution device (GPU, CPU, or other accelerators), and the frozen model to use. There are two ways to access this library. One is by linking it with an existing C++ application, using the two provided standalone applications as a usage example. The second one is to use the provided ROS node, which already takes care of everything needed to do the inference, such as de-bayering the input images, resizing, and publishing the mask topics, so that no coding is needed. List. 9.5 contains an example of how to build a small "main.cpp" application to perform semantic segmentation on an image from disk using our C++ library.

Table 9.1: Pixel-wise metrics for sample architectures.

Dataset	Arch.	Input Size	#Param	#Ops.	mIoU	mAcc.
Cityscapes	ERFNet	1024x512	1.8M	66B	62.8%	92.7%
	Mobilenets	768x384	6.9M	72B	63.5%	93.7%
	Inception	768x384	4.3M	47B	66.4%	94.1%
COCO Persons	Inception	640x480	4.3M	48B	87.1%	97.8%
		320x240		12B	83.4%	96.9%
Synthia	ERFNet	512x384	1.8M	24B	64.1%	92.3%
		960x720		85B	71.3%	95.2%
Crop-Weed	ERFNet	512x384	1.1M	9B	80.1%	98.5%

9.4 Sample Use Cases Shipped with Bonnet

In order to show the capabilities of Bonnet, we provide three sample architectures focusing on real-time inference. The three models included are based on ERFNet [151], InceptionV3 [170], and MobilenetsV2 [159], with minor modifications, which running the architectures in TensorRT. This framework supports a large subset of all TensorFlow operations and makes the networks much faster to run, as we show in Table 9.2.

Table 9.1 shows the performance of the sample architectures on four diverse and challenging datasets, two for scene parsing, one for people segmentation, and one for precision agriculture purposes, for which we provide the trained weights. Because each problem presents a different level of difficulty and uses images of a different aspect ratio, we show the performance of the model for different number of parameters and number of operations by varying the number of kernels of each layer of the base architecture and the size of the input.

Since Bonnet is meant to serve as a general starting point to implement different architectures, we advise referring to the code in order to have an up-to-date measure of the latest architecture design performances.

Table 9.2 shows the runtime of the ERFNet based model, with varying complexity and input size. It shows how much the inference time can be improved by using custom accelerators for the available commercial hardware. This further supports the importance of allowing the user to transparently benefit from its use with no extra coding effort, as well as providing a modular C++ backend that allows the support of other backends as they become available.

Table 9.2: Mean runtime of the ERFNet-based architecture for different datasets, sizes, and backends.

Dataset	Input Size	Back-end	GTX1080Ti	Jetson TX2
Cityscapes	512x256	TensorFlow	19ms (52 FPS)	170ms (6 FPS)
		TensorRT	10ms (100 FPS)	89ms (11 FPS)
	1024x512	TensorFlow	71ms (14 FPS)	585ms (2 FPS)
		TensorRT	33ms (30 FPS)	245ms (4 FPS)
COCO Persons	640x480	TensorFlow	27ms (37 FPS)	321ms (3 FPS)
		TensorRT	15ms (65 FPS)	128ms (8 FPS)
	320x240	TensorFlow	21ms (47 FPS)	200ms (5 FPS)
		TensorRT	7ms (142 FPS)	80ms (14 FPS)
Synthia	512x384	TensorFlow	20ms (50 FPS)	223ms (4 FPS)
		TensorRT	11ms (100 FPS)	127ms (8 FPS)
	960x720	TensorFlow	61ms (16 FPS)	673ms (1 FPS)
		TensorRT	27ms (37 FPS)	362ms (3 FPS)
Crop-Weed	512x384	TensorFlow	9ms (111 FPS)	132ms (8 FPS)
		TensorRT	4ms (250 FPS)	99ms (10 FPS)

9.5 Conclusion

In this chapter, we presented Bonnet, an open-source semantic segmentation training and deployment tool for robotics research. Bonnet eases the integration of semantic segmentation methods for robotics. It provides a stable interface allowing the community to better collaborate, add different datasets and network architectures, and share implementation efforts as well as pre-trained models. This tool has sped up the deployment of semantic segmentation CNNs on research robotics platforms in our lab, and for our students. We provide three sample architectures that operate at frame-rate and include pre-trained weights for diverse and challenging datasets for the robotics community.

9.6 Outlook: Extension to Other Tasks

Since the release of Bonnet, we have switched our internal developments to PyTorch [127], since its dynamic graph software philosophy allows for a more "pythonic" execution of the training as well as the deployment. This makes it more flexible for non-standard tasks that diverge from standard fully-feedforward networks, such networks dealing with variably-sized inputs and outputs, sequence data of different sequence length, or containing recursion and/or loops. Long after our development switch, Tensorflow 2.0 was released, also allowing for imperative execution, but at the time we had already been using PyTorch for a year. Since the switch made us re-write our entire pipeline, we decided to extend the

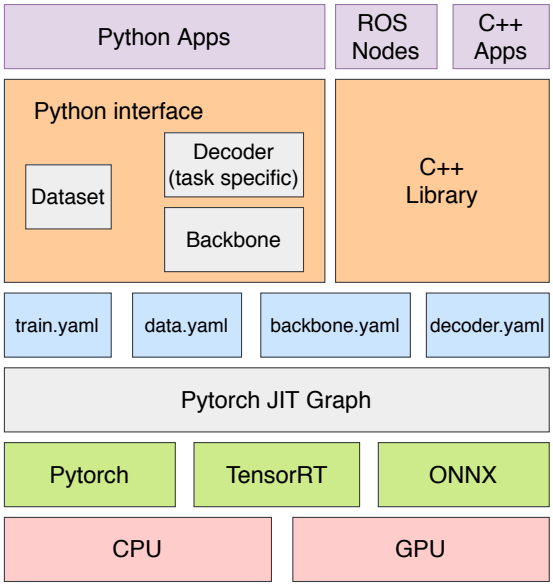

Figure 9.6: Abstraction of the new codebase. Python interface is used for training and graph definition for all tasks, and C++ library can use a trained graph and infer semantic segmentation in any running application, either linking it or by using the ROS node. Both interfaces communicate through the four configuration files in *yaml* format and the trained model weights. The difference with Bonnet is that the Backbone can be shared between different tasks, and pretrained on Imagenet [34] as it is standard transfer-learning practice.

framework to more tasks, rather than just semantic segmentation. In Bonnetal, which is our latest deep learning framework, we exploit the fact that CNN features learned for the classification task on large datasets tend to translate very well to other tasks, such as object detection, semantic and instance segmentation, etc [143]. To this end, we generalize the modular pipeline depicted in Figure 9.3 to follow the design in Figure 9.6

The new software framework also provides a deployment pipeline for all tasks implemented, both as a C++ library as well as in ROS nodes. This framework can be found at: `http://github.com/PRBonn/bonnetal`.

Chapter 10

Example Applications

Throughout this thesis, we have argued that semantic scene understanding has the potential to help multiple tasks in robotics, such as manipulation, obstacle detection and avoidance, remote monitoring, localization and mapping, path planning, precision agriculture, etc. In this chapter, we focus on particular use cases of research using the methods presented in this thesis to solve real-world problems. The approaches presented in this chapter use either Bonnet or Bonnetal if they use images as input, which was presented in Chapter 9. On the other hand, if the approach being depicted uses LiDAR point clouds as input, the open-source Lidar-Bonnetal pipeline to perform inference of semantics for LiDAR, presented in Chapter 7 is used.

In the following sections we give an overview of the task and how semantics extracted from one of our approaches were able to help achieve the goal in a better, or a faster way (or both). These examples applications include:

- precision agriculture, in Section 10.1
- path planning, in Section 10.2
- simultaneous localization and mapping (SLAM) using RGBD sensors, in Section 10.3, and
- SLAM using LiDAR sensors, in Section 10.4.

Figure 10.1: Berry detection and counting in wineyards. Left: Input RGB image from DSLR. Right: Overlay of semantic mask of regions containing berries from Bonnetal.

10.1 Berry Counting

Yield estimation and forecasting are of special interest in the field of grapevine breeding and viticulture. The number of harvested berries per plant is strongly correlated with the resulting quality. Therefore, early yield forecasting can enable a focused thinning of berries to ensure a high-quality end product. In this section, we present an effort to perform berry counting [190] lead by Laura Zabawa as a part of her Ph.D. work at Heiner Kuhlmann's group, and in which Bonnetal is a key component. We only provide a brief overview to highlight how our semantics helped, and more information can be found in the relevant publication [190].

Traditionally yield estimation is done by extrapolating from a small sample size and by utilizing historic data. Moreover, it needs to be carried out by skilled experts with much experience in this field. Berry detection in images offers a cheap, fast, and non-invasive alternative to the otherwise time-consuming and subjective on-site analysis by experts. We apply fully convolutional neural networks on images acquired with the Phenoliner, a field phenotyping platform. We count single berries in images to avoid the error-prone detection of grapevine clusters. Clusters are often overlapping and can vary a lot in the size which makes the reliable detection of them difficult. We address especially the detection of white grapes directly in the vineyard. The detection of single berries is formulated as a classification task with three classes, namely "berry", "edge" and "background". A connected component algorithm is applied to determine the number of berries in one image. We compare the automatically counted number of berries with the manually detected berries in 60 images showing Riesling plants in vertical shoot positioned trellis (VSP) and semi minimal pruned hedges (SMPH). We are able to detect berries correctly within the VSP system with an accuracy of 94.0% and for the SMPH system with 85.6%. Figure 10.1 shows an example of how Bonnetal is used to perform a pre-segmentation of the berries.

Figure 10.2: Room parsing for efficient path planning in children-inhabitated environments. Left: Humanoid robot using information from semantics to plan. Right: Camera view with semantic overlay of different toys in the scene from Bonnet.

10.2 Path Planning

Humanoid robots are often supposed to share their workspace with humans and thus have to deal with objects used by humans in their everyday life. In this section, we present our novel approach [145] to humanoid navigation through cluttered environments, which exploits knowledge about different obstacle classes to decide how to deal with obstacles and selects appropriate robot actions. This effort was led by Peter Regier as a part of his doctorate work at Maren Bennewitz's group, and Bonnet was a key component for training and inferring the semantic segmentation CNN online with edge-compute hardware. We only provide a brief overview to highlight how our semantics helped, and more information can be found in the relevant publications [145, 146].

To classify objects from RGB images and decide whether an obstacle can be overcome by the robot with a corresponding action, e.g., by pushing or carrying it aside or stepping over or onto it, see Figure 10.2, we train and exploit a CNN from Bonnet. Based on associated action costs of the observed objects in the scene, we compute a cost grid containing newly observed objects in addition to static obstacles on which a 2D path can be efficiently planned. This path encodes the necessary actions that need to be carried out by the robot to reach the goal. Using our CNN the robot can robustly classify the observed obstacles into the different classes and decide on suitable actions to find efficient solution paths. Our system finds paths also through regions where traditional motion planning methods are not able to calculate a solution or require substantially more time. We finally implemented the framework in ROS and tested it in various scenarios with a Nao robot as well as in simulation with the REEM-C robot. Furthermore, the labels for training the CNN using Bonnet were scraped from the internet querying images of toys with alpha background, which allowed us to generate a semi-synthetic dataset of 10 000 semantic segmentation images which worked well enough for the affordance estimation task in the real world.

Figure 10.3: Person segmentation for RGBD SLAM. Left: Semantic overlay of image areas containing people from Bonnet. Right: Result of the static reconstruction of the environment.

10.3 RGBD Localization and Mapping

Most approaches to RGBD simultaneous localization and mapping assume, as a part of their set of heuristics about the world, that the environment surrounding the robot is static, which is often violated in the real world. The current trend in the state of the art is to identify specific categories of dynamic elements using CNNs, and eliminate them from the observation of the environment. In this work, led by Emanuele Palazzolo as a part of his doctoral thesis [124], we compared purely geometric approaches to perform the static vs. dynamic segmentation and the aforementioned CNN-based one. Therefore, we implemented an algorithm based on Dr. Palazzolo's SLAM system in tandem with semantic segmentation of people using a pre-trained people segmentation CNN from Bonnet's library of pre-trained models.

Given that the people vs. background semantic segmentor is included in Bonnet, and that the SLAM system used CUDA, the included TensorRT accelerator and library access within Bonnet was a perfect fit for a quick implementation. With little more work than the one depicted in List. 9.5 in order to exploit the depth channel available to improve the segmentation, the SLAM system was able to consume solely the static areas of the image.

The experiments suggested that, indeed, the results of the approach are significantly better than consuming the entirety of the frame which also includes dynamic objects, especially when these occupy a large portion of the input frames. It is important to notice, however, that this is only true if all of the dynamic objects are both part of the label ontology for which CNN was trained (in this case just people), as well as accurately segmented. This leads to two types of problems. First, it does not address moving objects outside the label ontology for which the CNN was trained. In this work, this was solved by adding a geometric approach using residuals to extract unrecognized dynamics, which is a key contribution of Dr. Palazzolo's dissertation. Second, just because an object is potentially-dynamic does not mean it is not useful for registration, should the object be still for the entirety of the sequence, which we address in Section 10.4.

Figure 10.4: Results of a semantic mapping approach [27] using the semantic predictions from RangeNet++ in Lidar-Bonnetal. Left: Semantic mapping using all classes, which generates blurred, elongated objects. Right: Semantic mapping using only non-movable classes, which also removes non-moving objects just because they are potentially movable. This hurts the registration process. Middle: Our approach. The integration of the movable classes is down-weighed, as well as its contribution to the registration cost, resulting in the best model.

10.4 Semantically-augmented LiDAR Odometry

When driving in a real-world environment, besides geometric information about the mapped environment, the semantics of the environment play a key role if we want to enable intelligent navigation behaviors. In most realistic environments, this task is particularly complicated due to dynamics caused by moving objects, which can corrupt the mapping step or derail localization. In this section, we highlight the extension of a recently published surfel-based mapping approach exploiting three-dimensional laser range scans by integrating semantic information to facilitate the odometry estimation and mapping process. This effort was led by Xieyuanli Chen resulting in our approach SuMa++ approach [27]. In this work, we extracted the semantic information using a RangeNet++ model directly inferring in spherically projected LiDAR sweeps. This computed semantic segmentation results in point-wise labels for the whole scan, allowing us to build a semantically-enriched map with labeled surfels. This semantic map enables us to reliably filter moving objects, but also improve the projective scan matching via semantic constraints. This problem was already mentioned in Section 10.3 and its effect is shown in Figure 10.4 - left. In this work, instead of simply removing all potentially movable objects, which leads to a map losing a lot of useful texture for registration as in Figure 10.4 - right, we use the semantic labels to change the integration parameters of each surfel into the map, and its contribution to the ICP cost by using a map that assigns a lower weight to potentially moving classes. This allows us to use their texture if they are static, while still removing the dynamic ones from both the map, as well as the registration process, in the same way a robust kernel would, shown in the Figure 10.4 - middle.

Our experimental evaluation on challenging highways sequences from KITTI dataset with very few static structures and a large amount of moving cars show the advantage of our semantic SLAM approach in comparison to a purely geometric, state-of-the-art approach. For more details about this approach, refer to its relevant publication [27].

10.5 Conclusion

By showing these four rather diverse examples of real, practical use-cases of the predictions obtained by the models in Bonnet, Bonnetal, and LiDAR-Bonnetal, we have showcased real-world examples of all our semantic pipelines in action, being exploited downstream by other researchers in different fields. We strongly believe that the reproducibility and reusability of our research is key to keeping the field going forward at a steady pace, and thus, to make better science. This is why we strongly stand by our decision to open-source our models *in a stable, easy-to-use manner*. This means, open-sourcing the code as more than just a collection of throw-away scripts *just because you have to*, but rather as part of a stable framework that other practitioners can plug and play into their working pipelines. Naturally, this is likely too ambitious an enterprise, and our desire for full reusability is probably not fulfilled 100%, but as researchers and scientists, we need to strive to take the field in this direction. This becomes increasingly clear as while writing the last lines of this doctoral thesis, and thinking about the new generations of doctoral students in our laboratory who will be responsible for maintaining the software code-bases left behind. Bonnet, Bonnetal, and LiDAR-Bonnetal have incurred a significant amount of effort in their development, in order to make it easily accessible to other researchers, as well as to our undergraduate and graduate students, and we believe that, albeit not perfect, they provide a significant step in the right direction.

Chapter 11

Conclusion

As mobile robots become a part of our everyday lives, it is key for them to have an accurate understanding of their surroundings in order to operate safely and efficiently. This is true for the seemingly harmless cleaning robots already mopping and dusting away in our houses, to the scarier self-driving vehicles that are soon to be populating our city streets and highways, as well as for the agricultural robots that will handle our food. One thing that all of these systems have in common is that in order to operate safely they require the acquisition of this understanding in a fast and accurate manner, and using on-device processing, in order to reduce the latency between changes in the environment and action execution.

In this thesis, we have presented a number of approaches to achieve this level of required efficiency in multiple domains, ranging from the task of recognizing weeds in crop fields for precision agriculture, as well as analyzing driving scenes to move around in our cities autonomously.

In Part I we presented three approaches to semantic scene understanding using camera images. All of them exploit background knowledge about the application domain in order to reduce the amount of computation needed for the solution of the task. In Chapter 3 we presented an approach that allows us to perform selective weeding in crop fields, by extracting regions of interest efficiently through an extra visual cue in the NIR spectrum, and classifying them into crops and weeds. This pre-segmentation step makes the approach fast to run when compared with traditional object detection pipelines which extract proposals either through selective search or through a separate region-proposal neural network. This is one of the ways in which domain expertise helps the robot's perception system achieve its goal efficiently, even when the regions are later classified by a neural network trained in an end-to-end fashion, with no feature design. However, this type of approach relies on an, albeit frequently used in precision agriculture, expensive to obtain extra NIR visual cue. In order to allow the robot to perform the recogni-

tion task with any low-cost, commercial RGB camera, in Chapter 4, we propose an RGB-only approach to the same task. Here, we replace the expensive infrared cue by a suite of different mappings of the RGB input design by an expert. We showed through thorough experiments that these extra mappings allow us to perform both the segmentation and the classification tasks jointly. Besides this, we also showed that these extra channels make the approach more robust to changes in the environment, even when inferring in a different country, increasing the generalization capabilities, and reducing the amount of new data required for re-training if at all necessary. Moving from our crop fields towards our urban environments, in Chapter 5 we propose an approach to jointly perform semantic as well as instance segmentation of road scenes, which also exploits background information for the sake of algorithmic efficiency. However, obtaining useful cues for urban environments is not as straight-forward as it is in crop fields, so in this case, we try to exploit the geometry of the scene. In order to do this, we use the fact that connected regions in the image which are neighboring and have similar texture likely belong to the same object. This way, we are able to pre-segment our input into a lower-dimensional grid, without losing the accuracy in the boundaries of the prediction, making our approach just as accurate as its general counterpart, but several times faster to run. As we have mentioned before, all of these types of algorithmic decisions made were not just with the goal of making our approaches more accurate. Sometimes, as is the case with Chapter 5, we aim to achieve roughly the same level of accuracy as the state-of-the-art approaches, but running much faster, allowing us to deploy the algorithms in real mobile robots running in real environments, rather than just reporting numbers on a table.

In Part II, we presented approaches that tackle similar problems, albeit using a different sensor modality. Here, we focus on the efficient semantic scene understanding of LiDAR-obtained point clouds, which is of current and future relevance in the context of autonomous driving. First, in Chapter 6 we present our large-scale semantic and panoptic segmentation dataset and benchmark evaluation for LiDAR point clouds, called SemanticKITTI. This large scale dataset, along with its evaluation server, serves the main purpose of providing a platform in which researchers in the community can, not only develop their deep learning approaches but also compare the results in a standardized way, with a proper hidden test set. In the following chapters, we present initial approaches to tackle both these tasks in an efficient manner, so that they can run on hardware that can be easily fitted to a car. In Chapter 7 we propose an approach that overcomes the variable sparsity problem present in LiDAR scans by using a range image-like projection of the point cloud, for the task of semantic segmentation. This approach succeeds where others fail, mostly due to the fact that it does not require the point clouds to be sampled uniformly from a surface, and because its

2-dimensional nature makes it efficient, allowing us to use larger models. However, this does not come without drawbacks. The 2-dimensional representation of the data causes blurry predictions and shared frustums to bleed wrong labels into the original point cloud. Therefore, we proposed an efficient re-projection algorithm based on an approximate, GPU-enabled, k-nearest neighbor search, that runs in a couple of ms per scan. Building on this work and exploiting the panoptic labels included in SemanticKITTI, in Chapter 8 we also presented the first two baselines to the panoptic segmentation of LiDAR point clouds, as well as a novel, single-stage approach achieving comparable results, albeit at a fraction of the computational cost. As we mentioned already, this is of utmost importance in robotics, and particularly for autonomous vehicles that need to react quickly to changes in the scene while driving at high speeds.

Finally, in Part III we presented a software design paradigm for semantic scene understanding which we follow internally for our development, and which serves as a starting point for future generations to develop their own solutions. Furthermore, we also released three open-source frameworks for semantic scene understanding which follow this design philosophy.

11.1 Open Source Contributions

Besides our many peer-reviewd paper publications, many of the algorithms presented in this thesis can be accessed in the open-source frameworks we released:

- The work in Chapter 4 is available as a part of Bonnet, which is also the framework presented in Chapter 9. This framework is available online at: `http://github.com/PRBonn/bonnet`,
- The work in Chapter 5 is a part of Bonnetal, which is also presented as Section 9.6 as Bonnetal, and is available at: `http://github.com/PRBonn/bonnetal`,
- The dataset presented in Chapter 6, along with both benchmarks is available online at: `http://semantic-kitti.org`,
- The API to access said dataset, visualize the labels, and evaluate the performance on the validation set before submission to the benchmark is available online at: `http://github.com/PRBonn/semantic-kitti-api`,
- The code for Chapter 7 and Chapter 8 is on our last open-source framework: `http://github.com/PRBonn/lidar-bonnetal`.

11.2 Future Work

There is no better way to create a *future work* section than to state the things for which there was a lot of will, but not a lot of time to do, and which we would love to see realized.

First of all, the approaches presented in this thesis were *supervised learning* approaches. This means that they all required human-labeled data in order to be trained. One of the most interesting research avenues that are yet to be explored in depth is how *self-supervised learning* can be used for the applications that were described in this thesis. Although multiple works have been proposed that exploit redundancies in the data (e.g., predicting the depth from RGB images, and training with data extracted from an RGBD sensor), not much work has been done in the extraction of semantics. For this, the sequential data provided in Chapter 8 is a very good starting point, since this sequentiality can provide much of the needed redundancy to make self-supervised learning work in the context of LiDAR and autonomous driving. Going fully self-supervised, or even completely un-supervised, however, is clearly another long-term research endeavor. Regardless of this, labeled data is required for the validation of the approaches, so the dataset released in Chapter 6 will likely be relevant for many years to come, even as we make the transition towards self-supervision.

Second, one of the most important and immediate research avenues is the use of the mentioned *sequential* data to improve the results of the semantic and panoptic segmentation of LiDAR for autonomous driving. This is an area of untapped potential and where we suspect industry is ahead of the curve, given the massive amount of sequential data they collect, although some movement in that direction has started to happen [69]. In order to democratize this, we need university researchers tackling this problem, and providing this data was one of the ways in which we made our contribution to the field. Now it is time to actually put it to work.

Finally, one of the most worrying topics during the entire time we worked on the LiDAR data was the lack of transferability of the CNN-extracted features to be used with a different LiDAR sensor, when trained with one particular setup. This was address in our work under submission [83], where we adapted the features to a new sensor domain with a lower resolution and different field of view and position in the car in an unsupervised way, but it is likely an area of untapped potential that is waiting to be awakened, and where quick progress can likely be made.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint*, 1603.04467v2, 2016.

[2] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012.

[3] I. Alonso, L. Riazuelo, L. Montesano, and A.C. Murillo. 3D-MiniNet: Learning a 2D Representation from Point Clouds for Fast and Efficient 3D LIDAR Semantic Segmentation. *arXiv preprint*, 2002.10893v2, 2020.

[4] A. Armagan, M. Hirzer, and V. Lepetit. Semantic segmentation for 3d localization in urban environments. In *2017 Joint Urban Remote Sensing Event (JURSE)*, pages 1–4, 2017.

[5] I. Armeni, A. Sax, A. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *arXiv preprint*, 1702.01105, 2017.

[6] N. Atanasov, M. Zhu, K. Daniilidis, and G.J. Pappas. Localization from semantic observations via the matrix permanent. *Intl. Journal of Robotics Research (IJRR)*, 35(1-3):73–99, 2016.

[7] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, 2017.

[8] M. Bai and R. Urtasun. Deep watershed transform for instance segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[9] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[10] J. Behley, A. Milioto, and C. Stachniss. A Benchmark for LiDAR-based Panoptic Segmentation based on KITTI. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[11] J. Behley, A. Milioto, and Cyrill Stachniss. A Benchmark for LiDAR-based Panoptic Segmentation based on KITTI. *arXiv preprint*, 2020.

[12] J. Behley and C. Stachniss. Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments. In *Proc. of Robotics: Science and Systems (RSS)*, 2018.

[13] J. Behley, V. Steinhage, and A.B. Cremers. Performance of Histogram Descriptors for the Classification of 3D Laser Range Data in Urban Environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2012.

[14] J. Behley, V. Steinhage, and A.B. Cremers. Laser-based Segment Classification Using a Mixture of Bag-of-Words. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[15] N. Blodow, L. C. Goron, Z. C. Marton, D. Pangercic, T. Rühr, M. Tenorth, and M. Beetz. Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4263–4270, 2011.

[16] B. De Brabandere, D. Neven, and L. Van Gool. Semantic instance segmentation with a discriminative loss function. In *Deep Learning for Robotic Vision workshop, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[17] G.J. Brostow, J. Fauqueur, and R. Cipolla. Semantic Object Classes in Video: A High-Definition Ground Truth Database. *Pattern Recognition Letters*, 2008.

[18] H. Caesar, V. Bankiti, A.H. Lang, S. Vora, V.E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A multimodal dataset for autonomous driving. *arXiv preprint*, 1903.11027v1, 2019.

[19] G. Cerutti, L. Tougne, J. Mille, A. Vacavant, and D. Coquin. A model-based approach for compound leaves understanding and identification. In *Proc. of the IEEE Intl. Conf. on Image Processing (ICIP)*, pages 1471–1475, 2013.

[20] A. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], 2015.

[21] M.F. Chang, J.W. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays. Argoverse: 3D Tracking and Forecasting with Rich Maps. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[22] N. Chebrolu, P. Lottes, A. Schaefer, W. Winterhalter, W. Burgard, and C. Stachniss. Agricultural robot dataset for plant classification, localization and mapping on sugar beet fields. *Intl. Journal of Robotics Research (IJRR)*, 2017.

[23] L. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv preprint*, 1706.05587v3, 2017.

[24] S.W. Chen, S. Skandan, S. Dcunha, J. Das, C. Qu, C.J. Taylor, and V. Kumar. Counting Apples and Oranges with Deep Learning: A Data Driven Approach. *IEEE Robotics and Automation Letters (RA-L)*, 2017.

[25] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint*, abs/1604.06174, 2016.

[26] X. Chen, T. Laebe, A. Milioto, T. Roehling, O. Vysotska, A. Haag, J. Behley, and C. Stachniss. OverlapNet: Loop Closing for LiDAR-based SLAM. In *Proc. of Robotics: Science and Systems (RSS)*, 2020.

[27] X. Chen, A. Milioto, E. Palazzolo, P. Giguère, J. Behley, and C. Stachniss. SuMa++: Efficient LiDAR-based Semantic SLAM. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[28] Z. Chen, V. Badrinarayanan, C.Y. Lee, and A. Rabinovich. GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks. *arXiv preprint*, 1711.02257v4, 2017.

[29] B. Cheng, M.D. Collins, Y. Zhu, T. Liu, T.S. Huang, H. Adam, and L. Chen. Panoptic-DeepLab. In *Proc. of the ICCV Workshop: Joint COCO and Mapillary Recognition Challlenge Workshop*, volume 1910.04751v3, 2019.

[30] M. Cordts, S. Mohamed Omran, Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[31] T. Cortinhal, G. Tzelepis, and E.E. Aksoy. SalsaNext: Fast Semantic Segmentation of LiDAR Point Clouds for Autonomous Driving. *arXiv preprint*, 2003.03653v1, 2020.

[32] A. Dai, A.X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[33] A. Dai, D. Ritchie, M. Bokeloh, S. Reed, J. Sturm, and M. Nießner. ScanComplete: Large-Scale Scene Completion and Semantic Segmentation for 3D Scans. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

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

[35] M. Di Cicco, C. Potena, G. Grisetti, and A. Pretto. Automatic Model Based Dataset Generation for Fast and Accurate Crop and Weeds Detection. *arXiv preprint*, 1612.03019v3, 2016.

[36] R. Drouilly, P. Rives, and B. Morisset. Semantic representation for navigation in large-scale environments. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1106–1111, 2015.

[37] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C.C. Lerma. SegMatch: Segment Based Place Recognition in 3D Point Clouds. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[38] E. Elhariri, N. El-Bendary, and A. E. Hassanien. Plant classification system based on leaf features. In *Proc. of Computer Engineering Systems (ICCES)*, pages 271–276, 2014.

[39] F. Engelmann, T. Kontogianni, J. Schult, and B. Leibe. Know What Your Neighbors Do: 3D Semantic Segmentation of Point Clouds. *arXiv preprint*, 1810.01151, 2018.

[40] M. Everingham, S. Eslami, L. van Gool, C. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes Challenge – a Retrospective. *International Journal on Computer Vision (IJCV)*, 111(1):98–136, 2015.

[41] A. Fathi, Z. Wojna, V. Rathod, P. Wang, H.O. Song, S. Guadarrama, and K.P. Murphy. Semantic Instance Segmentation via Deep Metric Learning. *arXiv preprint*, 1703.10277v1, 2017.

[42] K. Franz, R. Roscher, A. Milioto, S. Wenzel, and J. Kusche. Ocean eddy identification and tracking using neural networks. In *Proc. of the IEEE Int. Geoscience and Remote Sensing Symposium (IGARSS)*, 2018.

[43] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual Worlds as Proxy for Multi-Object Tracking Analysis. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[44] F.J. Garcia-Ruiz, D. Wulfsohn, and J. Rasmussen. Sugar beet (beta vulgaris l.) and thistle (cirsium arvensis l.) discrimination based on field spectral data. *Biosystems Engineering*, 139:1–15, 2015.

[45] S. Garg, N. Snderhauf, and M.J. Milford. Don't look back: Robustifying place categorization for viewpoint and condition-invariant place recognition. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[46] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.

[47] J. Geipel, J. Link, and W. Claupein. Combined spectral and spatial modeling of corn yield based on aerial images and crop surface models acquired with an unmanned aircraft system. *Remote Sensing*, 6(11):10335, 2014.

[48] J. Geyer, Y. Kassahun, M. Mahmudi, X. Ricou, R. Durgesh, A.S. Chung, L. Hauswald, V.H. Pham, M. Mühlegg, S. Dorn, T. Fernandez, M. Jänicke, S. Mirashi, C. Savani, M. Sturm, O. Vorobiov, M. Oelker, S. Garreis, and P. Schuberth. A2D2: AEV Autonomous Driving Dataset. `http://www.a2d2.audi`, 2019.

[49] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[50] P. Goyal, P. Dollar, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arxiv*, 1706.02677v1, 2017.

[51] B. Graham, M. Engelcke, and L. van der Maaten. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[52] F. Groh, P. Wieschollek, and H. Lensch. Flex-Convolution (Million-Scale Pointcloud Learning Beyond Grid-Worlds). In *Proc. of the Asian Conf. on Computer Vision (ACCV)*, Dezember 2018.

[53] J. M. Guerrero, G. Pajares, M. Montalvo, J. Romeo, and M. Guijarro. Support vector machines for crop/weeds identification in maize fields. *Expert Systems with Applications*, 39(12):11149 – 11155, 2012.

[54] W. Guo, U.K. Rage, and S. Ninomiya. Illumination invariant segmentation of vegetation for time series wheat images based on decision tree model. *Computers and Electronics in Agriculture*, 96:58–66, 2013.

[55] T. Hackel, N. Savinov, L. Ladicky, Jan D. Wegner, K. Schindler, and M. Pollefeys. SEMANTIC3D.NET: A new large-scale point cloud classification benchmark. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume IV-1-W1, pages 91–98, 2017.

[56] D. Hall, C.S. McCool, F. Dayoub, N. Sunderhauf, and B. Upcroft. Evaluation of features for leaf classification in challenging conditions. In *Proc. of the IEEE Winter Conf. on Applications of Computer Vision (WACV)*, pages 797–804, Jan 2015.

[57] E. Hamuda, M. Glavin, and E. Jones. A survey of image processing techniques for plant extraction and segmentation in the field. *Computers and Electronics in Agriculture*, 125:184–199, 2016.

[58] S. Haug, A. Michaels, P. Biber, and J. Ostermann. Plant classification system for crop / weed discrimination without segmentation. In *Proc. of the IEEE Winter Conf. on Applications of Computer Vision (WACV)*, pages 1142–1149, 2014.

[59] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2017.

[60] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[61] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint*, abs/1207.0580, 2012.

[62] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[63] J. Hou, A. Dai, and M. Niessner. 3D-SIS: 3D Semantic Instance Segmentation of RGB-D Scans. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[64] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q.V. Le, and H. Adam. Searching for MobileNetV3. *arXiv preprint*, 1905.02244v5, 2019.

[65] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. *arXiv preprint*, 1911.11236v3, 2019.

[66] B. Hua, Q. Pham, D. Nguyen, M. Tran, L. Yu, and S. Yeung. SceneNN: A Scene Meshes Dataset with aNNotations. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, 2016.

[67] B. Hua, M. Tran, and S. Yeung. Pointwise Convolutional Neural Networks. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[68] Jing Huang and Suya You. Point Cloud Labeling using 3D Convolutional Neural Network. In *Proc. of the Internation Conference on Pattern Recognition (ICPR)*, 2016.

[69] J.V. Hurtado, R. Mohan, W. Burgard, and A. Valada. MOPT: Multi-Object Panoptic Tracking. *arXiv preprint*, 2004.08189v2, 2020.

[70] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint*, abs/1502.03167, 2015.

[71] V. Jampani, M. Kiefel, and P.V. Gehler. Learning Sparse High Dimensional Filters: Image Filtering, Dense CRFs and Bilateral Neural Networks. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[72] M. Jiang, Y. Wu, and C. Lu. PointSIFT: A SIFT-like Network Module for 3D Point Cloud Semantic Segmentation. *arXiv preprint*, 1807.00652, 2018.

[73] G.C. Devol Jr. U.S. Patent US2988237A, June 1961.

[74] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Lyft Level 5 AV Dataset 2019. https://level5.lyft.com/dataset/, 2019.

[75] R. Khanna, M. Möller, J. Pfeifer, F. Liebisch, A. Walter, and R. Siegwart. Beyond point clouds - 3d mapping and field parameter measurements using uavs. In *Proc. of the IEEE Conf. on Emerging Technologies Factory Automation (ETFA)*, pages 1–4, 2015.

[76] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár. Panoptic Segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[77] R. Klukov and V. Lempitsky. Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2017.

[78] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[79] N. Kumar, P. N. Belhumeur, A. Biswas, D. W.Jacobs, W. J. Kress, I. López, and J. V. B. Soares. Leafsnap: A computer vision system for automatic plant species identification. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2012.

[80] J.D. Lafferty, A. McCallum, and F.C.N. Pereira. Conditional Random Fields: Probabilistic Models for Segmentating and Labeling Sequence Data. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2001.

[81] L. Landrieu and M. Simonovsky. Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[82] A.H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. PointPillars: Fast Encoders for Object Detection From Point Clouds. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[83] F. Langer, A. Milioto, A. Haag, J. Behley, and C. Stachniss. Domain Transfer for Semantic Segmentation of LiDAR Data using Deep Neural Networks. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[84] M. V. Latte, B. S. Anami, and Kuligod V. B. A combined color and texture features based methodology for recognition of crop field image. *Int. Journal of Signal Processing, Image Processing and Pattern Recognition*, 8(2):287–302, 2015.

[85] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521:436–444, 2015.

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

[87] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proc. of the IEEE*, pages 2278–2324, 1998.

[88] W. Li, S. Saeedi, J. McCormac, R. Clark, D. Tzoumanikas, Q. Ye, Y. Huang, R. Tang, and S. Leutenegger. InteriorNet: Mega-scale Multi-sensor Photo-realistic Indoor Scenes Dataset. In *Proc. of British Machine Vision Conference (BMVC)*, 2018.

[89] Y. Li, X. Chen, Z. Zhu, L. Xie, G. Huang, D. Du, and X. Wang. Attention-Guided Unified Network for Panoptic Segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[90] F. Liebisch, J. Pfeifer, R. Khanna, P. Lottes, C. Stachniss, T. Falck, S. Sander, R. Siegwart, A. Walter, and E. Galceran. Flourish – A robotic approach for automation in crop management. In *In Proc. of the Workshop für Computer-Bildanalyse und unbemannte autonom fliegende Systeme in der Landwirtschaft*, 2016.

[91] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, pages 740–755, 2014.

[92] T.Y. Lin, P. Goyal, R.B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv preprint*, abs/1708.02002, 2017.

[93] H. Liu, C. Peng, C. Yu, J. Wang, X. Liu, G. Yu, and W. Jiang. An End-To-End Network for Panoptic Segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[94] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[95] P. Lottes, J. Behley, N. Chebrolu, A. Milioto, and C. Stachniss. Joint Stem Detection and Crop-Weed Classification for Plant-specific Treatment in Precision Farming. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[96] P. Lottes, J. Behley, N. Chebrolu, A. Milioto, and C. Stachniss. Robust joint stem detection and crop-weed classification using image sequences for plant-specific treatment in precision farming. *Journal of Field Robotics (JFR)*, 37:20–34, 2020.

[97] P. Lottes, J. Behley, A. Milioto, and C. Stachniss. Fully convolutional networks with sequential information for robust crop and weed detection in precision farming. *IEEE Robotics and Automation Letters (RA-L)*, 3:3097–3104, 2018.

[98] P. Lottes, M. Höferlin, S. Sander, M. Müter, P. Schulze-Lammers, and C. Stachniss. An Effective Classification System for Separating Sugar Beets and Weeds for Precision Farming Applications. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2016.

[99] P. Lottes, M. Höferlin, S. Sander, M. Müter, P. Schulze-Lammers, and C. Stachniss. An effective classification system for separating sugar beets and weeds for precision farming applications. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2016.

[100] P. Lottes, M. Höferlin, S. Sander, and C. Stachniss. Effective vision-based classification for separating sugar beets and weeds for precision farming. *Journal of Field Robotics (JFR)*, 2016.

[101] P. Lottes, M. Höferlin, S. Sander, and C. Stachniss. Effective Vision-based Classification for Separating Sugar Beets and Weeds for Precision Farming. *Journal of Field Robotics (JFR)*, 34:1160–1178, 2017.

[102] P. Lottes, R. Khanna, J. Pfeifer, R. Siegwart, and C. Stachniss. Uav-based crop and weed classification for smart farming. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[103] P. Lottes and C. Stachniss. Semi-supervised online visual crop and weed classification in precision farming exploiting plant arrangement. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[104] D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[105] C.S. McCool, T. Perez, and B. Upcroft. Mixtures of Lightweight Deep Convolutional Neural Networks: Applied to Agricultural Robotics. *IEEE Robotics and Automation Letters (RA-L)*, 2017.

[106] J. McCormac, R. Clark, M. Bloesch, A.J. Davison, and S. Leutenegger. Fusion++: Volumetric Object-Level SLAM. *arXiv preprint*, 1808.08378v2, 2018.

[107] J. McCormac, A. Handa, S. Leutenegger, and A.J.Davison. SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation? In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2017.

[108] G.E. Meyer and J. Camargo-Neto. Verification of color vegetation indices for automated crop imaging applications. *Computers and Electronics in Agriculture*, 63(2):282–293, 2008.

[109] A. Milioto, J. Behley, C.S. McCool, and C. Stachniss. LiDAR Panoptic Segmentation for Autonomous Driving. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[110] A. Milioto, P. Lottes, and C. Stachniss. Real-time blob-wise sugar beets vs weeds classification for monitoring fields using convolutional neural networks. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2017.

[111] A. Milioto, P. Lottes, and C. Stachniss. Real-time Semantic Segmentation of Crop and Weed for Precision Agriculture Robots Leveraging Background Knowledge in CNNs. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[112] A. Milioto, L. Mandtler, and C. Stachniss. Fast Instance and Semantic Segmentation Exploiting Local Connectivity, Metric Learning, and One-Shot Detection for Robotics. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[113] A. Milioto and C. Stachniss. Bonnet: An Open-Source Training and Deployment Framework for Semantic Segmentation in Robotics using CNNs. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[114] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation. In *Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[115] R. Mohan and A. Valada. EfficientPS: Efficient Panoptic Segmentation. *arXiv preprint*, 2004.02307v2, 2020.

[116] A.K. Mortensen, M. Dyrmann, H. Karstoft, R. N. Jörgensen, and R. Gislum. Semantic Segmentation of Mixed Crops using Deep Convolutional Neural Network. In *Proc. of the International Conf. of Agricultural Engineering (CIGR)*, 2016.

[117] M. Muja and D. G. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, 36, 2014.

[118] D. Munoz, J. A. Bagnell, N. Vandapel, and M. Hebert. Contextual Classification with Functional Max-Margin Markov Networks. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[119] G. Neuhold, T. Ollmann, S. Rota Bulo, and P. Kontschieder. The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2017.

[120] D. Neven, B. De Brabandere, M. Proesmans, and L. Van Gool. Instance Segmentation by Jointly Optimizing Spatial Embeddings and Clustering Bandwidth. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[121] D. Neven, B.D. Brabandere, S. Georgoulis, M. Proesmans, and L.V. Gool. Fast Scene Understanding for Autonomous Driving. *arXiv preprint*, 1708.02550v1, 2017.

[122] National Congress of Argentina. Argentinean Law 26.871. Declaration of Mate as National Infusion, 2013.

[123] N. Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[124] E. Palazzolo. *Active 3D Reconstruction for Mobile Robots*. PhD thesis, University of Bonn, 2019.

[125] S.A Papert. The Summer Vision Project. 1966.

[126] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation. *arXiv preprint*, 1606.02147v1, 2016.

[127] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv preprint*, 1912.01703v1, 2019.

[128] C. Payer, D. Štern, T. Neff, H. Bischof, and M. Urschler. Instance Segmentation and Tracking with Cosine Embeddings and Recurrent Hourglass Networks. *arXiv preprint*, 1806.02070v3, 2018.

[129] M. Perez-Ortiz, J. M. Peña, P. A. Gutierrez, J. Torres-Sánchez, C. Hervás-Martínez, and F. López-Granados. A semi-supervised system for weed mapping in sunflower crops using unmanned aerial vehicles and a crop row detection method. *Applied Soft Computing*, 37:533 – 544, 2015.

[130] M. Perez-Ortiz, J. M. Peña, P. A. Gutierrez, J. Torres-Sánchez, C. Hervás-Martínez, and F. López-Granados. Selecting patterns and features for between- and within- crop-row weed mapping using uav-imagery. *Expert Systems with Applications*, 47:85 – 94, 2016.

[131] J. Pfeifer, R. Khanna, C. Dragos, M. Popovic, E. Galceran, N. Kirchgessner, A. Walter, R. Siegwart, and F. Liebisch. Towards automatic uav data interpretation for precision farming. *Proc. of the Int. Conf. of Agricultural Engineering (CIGR)*, 2016.

[132] Q.H. Pham, D.T. Nguyen, B.S. Hua, G. Roig, and S.K. Yeung. JSIS3D: Joint Semantic-Instance Segmentation of 3D Point Clouds With Multi-Task Pointwise Networks and Multi-Value Conditional Random Fields. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[133] M.P. Ponti. Segmentation of low-cost remote sensing images combining vegetation indices and mean shift. *IEEE Geoscience and Remote Sensing Letters*, 10(1):67–70, 2013.

[134] L. Porzi, S. Rota Bulo, A. Colovic, and P. Kontschieder. Seamless Scene Segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[135] J. Pöschmann, P. Neubert, S. Schubert, and P. Protzel. Synthesized semantic views for mobile robot localization. In *Proc. of the Europ. Conf. on Mobile Robotics (ECMR)*, pages 1–6, 2017.

[136] C. Potena, D. Nardi, and A. Pretto. Fast and accurate crop and weed identification with summarized train sets for precision agriculture. In *Proc. of Int. Conf. on Intelligent Autonomous Systems (IAS)*, 2016.

[137] C. Potena, D. Nardi, and A. Pretto. Fast and accurate crop and weed identification with summarized train sets for precision agriculture. In *Proc. of Int. Conf. on Intelligent Autonomous Systems (IAS)*, 2016.

[138] A. Pretto, S. Aravecchia, W. Burgard, N. Chebrolu, C. Dornhege, T. Falck, F. Fleckenstein, A. Fontenla, M. Imperoli, R. Khanna, F. Liebisch, P. Lottes, A. Milioto, D. Nardi, S. Nardi, J. Pfeifer, M. Popović, C. Potena, C. Pradalier, E. Rothacker-Feder, I. Sa, A. Schaefer, R. Siegwart, C. Stachniss, A. Walter, W. Winterhalter, X. Wu, and J. Nieto. Building an Aerial-Ground Robotics System for Precision Farming. *IEEE Robotics and Automation Magazine (RAM)*, 2020.

[139] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[140] Charles R. Qi, Or Litany, Kaiming He, and Leonidas J .Guibas. Deep Hough Voting for 3D Object Detection in Point Clouds. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2019.

[141] C.R. Qi, K. Yi, H. Su, and L. J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, 2017.

[142] F.M. De Rainville, A. Durand, F.A. Fortin, K. Tanguy, X. Maldague, B. Panneton, and M.J. Simard. Bayesian classification and unsupervised learning for isolating weeds in row crops. *Pattern Analysis and Applications*, 17(2):401–414, 2014.

[143] A.S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN Features off-the-shelf: an Astounding Baseline for Recognition. *arXiv preprint*, 1403.6382v3, 2014.

[144] J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement. *arXiv preprint*, 1804.02767v1, 2018.

[145] P. Regier, A. Milioto, P. Karkowski, C. Stachniss, and M. Bennewitz. Classifying Obstacles and Exploiting Knowledge about Classes for Efficient Humanoid Navigation. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (HUMANOIDS)*, 2018.

[146] P. Regier, A. Milioto, C. Stachniss, and M. Bennewitz. Classifying Obstacles and Exploiting Class Information for Humanoid Navigation Through Cluttered Environments. *The Int. Journal of Humanoid Robotics (IJHR)*, 17(02):2050013, 2020.

[147] C.Y. Ren, V.A. Prisacariu, and I.D. Reid. gSLICr: SLIC superpixels at over 250Hz. *arXiv preprint*, 1509.04232v1, 2015.

[148] M. Ren and R.S. Zemel. End-to-end instance segmentation and counting with recurrent attention. *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[149] D. Rethage, J. Wald, J. Sturm, N. Navab, and F. Tombari. Fully-Convolutional Point Networks for Large-Scale Point Clouds. *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2018.

[150] G. Riegler, A. Ulusoy, and A. Geiger. OctNet: Learning Deep 3D Representations at High Resolutions. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[151] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Trans. on Intelligent Transportation Systems (ITS)*, 19(1):263–272, 2018.

[152] B. Romera-Paredes and P.H.S. Torr. Recurrent instance segmentation. *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, abs/1511.08250, 2016.

[153] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[154] R.A. Rosu, P. Schütt, J. Quenzel, and S. Behnke. LatticeNet: Fast Point Cloud Segmentation Using Permutohedral Lattices. *arXiv preprint*, 1912.05905v2, 2019.

[155] J. W. Rouse, Jr., R. H. Haas, J. A. Schell, and D. W. Deering. Monitoring Vegetation Systems in the Great Plains with Erts. *NASA Special Publication*, 351:309, 1974.

[156] X. Roynard, J. Deschaud, and F. Goulette. Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *Intl. Journal of Robotics Research (IJRR)*, 37(6):545–557, 2018.

[157] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. *Learning Representations by Back-Propagating Errors*, page 696–699. MIT Press, Cambridge, MA, USA, 1988.

[158] P.A. Samuelson. *Microeconomics*. McGraw-Hill Education (ISE Editions), 2001.

[159] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arxiv*, 1801.04381v3, 2018.

[160] ScaleAI. *PandaSet: Public large-scale dataset for autonomous driving*, 2020.

[161] M. Schwarz, A. Milan, A.S. Periyasamy, and S. Behnke. Rgb-d object detection and semantic segmentation for autonomous manipulation in clutter. *Intl. Journal of Robotics Research (IJRR)*, 2017.

[162] R. Sheikh, A. Milioto, P. Lottes, C. Stachniss, M. Bennewitz, and T. Schultz. Gradient and log-based active learning for semantic segmentation of crop and weed for agricultural robots. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020.

[163] N. Silberman, D. Hoiem, P. Kohli, and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2012.

[164] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[165] B. Steder, G. Grisetti, and W. Burgard. Robust Place Recognition for 3D Range Data Based on Point Features. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2010.

[166] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M-H. Yang, and J. Kautz. SPLATNet: Sparse Lattice Networks for Point Cloud Processing. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[167] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. *arXiv preprint*, 1912.04838, 2019.

[168] N. Sünderhauf, F. Dayoub, S. McMahon, B. Talbot, R. Schulz, P.I. Corke, G. Wyeth, B. Upcroft, and M. Milford. Place categorization and semantic mapping on a mobile robot. *arXiv preprint*, abs/1507.02428, 2015.

[169] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint*, abs/1602.07261, 2016.

[170] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. *arXiv preprint*, 1512.00567v3, 2015.

[171] M. Tatarchenko, J. Park, V. Koltun, and Q-Y. Zhou. Tangent Convolutions for Dense Prediction in 3D. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[172] L. Tchapmi, C. Choy, I. Armeni, J. Gwak, and S. Savarese. SEGCloud: Semantic Segmentation of 3D Point Clouds. In *Proc. of the International Conference on 3D Vision (3DV)*, 2017.

[173] G. Te, W. Hu, Z. Guo, and A. Zheng. RGCNN: Regularized Graph CNN for Point Cloud Segmentation. *arXiv preprint*, 1806.02952, 2018.

[174] H. Thomas, C.R. Qi, J. Deschaud, B. Marcotegui, F. Goulette, and L.J. Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. *arXiv preprint*, 1904.08889v2, 2019.

[175] A. Torralba and A.A. Efros. Unbiased Look at Dataset Bias. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[176] J. Torres-Sanchez, F. López-Granados, and J.M. Peña. An automatic object-based method for optimal thresholding in uav images: Application

for vegetation detection in herbaceous crops. *Computers and Electronics in Agriculture*, 114:43 – 52, 2015.

[177] J. Torres-Sanchez, F. López-Granados, and J.M. Peña. An automatic object-based method for optimal thresholding in uav images: Application for vegetation detection in herbaceous crops. *Computers and Electronics in Agriculture*, 114:43 – 52, 2015.

[178] S. Wang, S. Suo, W. Ma, A. Pokrovsky, and R. Urtasun. Deep Parametric Continuous Convolutional Neural Networks. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[179] W. Wang, R. Yu, Q. Huang, and U. Neumann. SGPN: Similarity Group Proposal Network for 3D Point Cloud Instance Segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[180] X. Wang, S. Liu, X. Shen, C. Shen, and J. Jia. Associatively Segmenting Instances and Semantics in Point Clouds. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[181] X.F. Wang, D. Huang, J. Du, H. Xu, and L. Heutte. Classification of plant leaf images with complicated background. *Applied Mathematics and Computation*, 205:916–926, 2008.

[182] Zo. Wang and F. Lu. VoxSegNet: Volumetric CNNs for Semantic Part Segmentation of 3D Shapes. *arXiv preprint*, 1809.00226, 2018.

[183] A. Wendel and J.P. Underwood. Self-Supervised Weed Detection in Vegetable Crops Using Ground Based Hyperspectral Imaging. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2016.

[184] B. Wu, A. Wan, X. Yue, and K. Keutzer. SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[185] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer. SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud. *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[186] Y. Xiong, R. Liao, H. Zhao, R. Hu, M. Bai, E. Yumer, and R. Urtasun. UPSNet: A Unified Panoptic Segmentation Network. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[187] C. Xu, B. Wu, Z. Wang, W. Zhan, P. Vajda, K. Keutzer, and M. Tomizuka. SqueezeSegV3: Spatially-Adaptive Convolution for Efficient Point-Cloud Segmentation. *arXiv preprint*, 2004.01803v1, 2020.

[188] B. Yang, J. Wang, R. Clark, Q. Hu, S. Wang, A. Markham, and N. Trigoni. Learning Object Bounding Boxes for 3D Instance Segmentation on Point Clouds. In *Proc. of the Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[189] L. Yi, W. Zhao, H. Wang, M. Sung, and L. Guibas. GSPN: Generative Shape Proposal Network for 3D Instance Segmentation in Point Cloud. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[190] L. Zabawa, A. Kicherer, L. Klingbeil, A. Milioto, R. Topfer, H. Kuhlmann, and R. Roscher. Detection of single grapevine berries in images using fully convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[191] W. Zeng and T. Gevers. 3DContextNet: K-d Tree Guided Hierarchical Learning of Point Clouds Using Local and Global Contextual Cues. *arXiv preprint*, 1711.11379, 2017.

[192] J. Zhang and S. Singh. Visual-Lidar Odometry and Mapping: Low-Drift, Robust, and Fast. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2015.

[193] Y. Zhang, Z. Zhou, P. David, X. Yue, Z. Xi, B. Gong, and H. Foroosh. PolarNet: An Improved Grid Representation for Online LiDAR Point Clouds Semantic Segmentation. *arXiv preprint*, 2003.14032v2, 2020.

[194] C. Zhao, H. Hu, and D. Gu. Building a grid-point cloud-semantic map based on graph for the navigation of intelligent wheelchair. In *Proc. of the Intl. Conf. on Automation and Computing (ICAC)*, pages 1–7, 2015.

[195] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. *arXiv preprint*, abs/1612.01105, 2016.

List of Figures

List of Tables

List of Algorithms