DGK Veröffentlichungen der DGK

Ausschuss Geodäsie der Bayerischen Akademie der Wissenschaften

Reihe C | Dissertationen | Heft Nr. 889

Torben Peters

# Learning Multi-View 2D to 3D Label Transfer for Semi-Supervised Semantic Segmentation of Point Clouds

München 2022

Bayerische Akademie der Wissenschaften

ISSN 0065-5325 ISBN 978-3-7696-5301-4

Diese Arbeit ist gleichzeitig veröffentlicht in:
Wissenschaftliche Arbeiten der Fachrichtung Geodäsie und Geoinformatik der Leibniz Universität Hannover
ISSN 0174-1454, Nr. 380, Hannover 2022

DGK Veröffentlichungen der DGK

Ausschuss Geodäsie der Bayerischen Akademie der Wissenschaften

Reihe C Dissertationen Heft Nr. 889

# Learning Multi-View 2D to 3D Label Transfer for Semi-Supervised Semantic Segmentation of Point Clouds

Von der Fakultät für Bauingenieurwesen und Geodäsie
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades
Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation

von

M.Sc. Torben Peters

geboren am 07.03.1988 in Langenhagen, Deutschland

München 2022

Bayerische Akademie der Wissenschaften

ISSN 0065-5325 ISBN 978-3-7696-5301-4

Diese Arbeit ist gleichzeitig veröffentlicht in:
Wissenschaftliche Arbeiten der Fachrichtung Geodäsie und Geoinformatik der Leibniz Universität Hannover
ISSN 0174-1454, Nr. 380, Hannover 2022

**Adresse der DGK:**




Ausschuss Geodäsie der Bayerischen Akademie der Wissenschaften (DGK)

Alfons-Goppel-Straße 11 ● D – 80 539 München

Telefon +49 – 89 – 23 031 1113 ● Telefax +49 – 89 – 23 031 - 1283 / - 1100

e-mail post@dgk.badw.de ● http://www.dgk.badw.de

Prüfungskommission:

Vorsitzender: Prof. Dr.-Ing. Steffen Schön

Referent: apl. Prof. Dr.-Ing. Claus Brenner

Korreferenten: apl. Prof. Dr. techn. Franz Rottensteiner
Prof. Dr. Konrad Schindler

Tag der mündlichen Prüfung: 14.07.2021

© 2022 Bayerische Akademie der Wissenschaften, München

Alle Rechte vorbehalten. Ohne Genehmigung der Herausgeber ist es auch nicht gestattet,
die Veröffentlichung oder Teile daraus auf photomechanischem Wege (Photokopie, Mikrokopie) zu vervielfältigen

ISSN 0065-5325 ISBN 978-3-7696-5301-4

## Abstract

Semantic segmentation is an important task in computer vision to help machines gain a high-level understanding of the environment, similar to the human vision system. For example it is used in self-driving cars which are equipped with various sensors such as cameras and 3D laser scanners to gain a complete understanding of their environment. In recent years the field has been dominated by Deep Neural Networks (DNNs), which are notorious for requiring large amounts of training data. Creating these datasets is very time consuming and costly. Moreover, the datasets can only be applied to a specific type of sensor. The present work addresses this problem. It will be shown that knowledge from publicly available image datasets can be reused to minimize the labeling costs for 3D point clouds. For this purpose, the labels from classified images are transferred to 3D point clouds. To bridge the gap between sensor modalities, the geometric relationship of the sensors in a fully calibrated system is used. Due to various errors the naive label transfer can lead to a significant amount of incorrect class label assignments in 3D. Within the work the different reasons and possible solutions are shown in order to improve the label transfer.

First, Scanstrip Network (SNet) is presented. The network learns to correct wrong class assignments in 3D point clouds and implicitly considers different sources of errors. It is trained in a supervised manner and only on a small amount of data. The simple but effective network design achieves an mean Intersection over Union (mIoU) of 0.67 as opposed to the baseline value of 0.48, outperforming similar and even state-of-the-art networks. These results are further improved by training SNet in a semi-supervised manner. For this, large amounts of automatically generated labels are used for pretraining, allowing the network to achieve a mIoU of 0.71.

One problem at the beginning of the label transfer is classification errors in images and wrong 2D pixels to 3D point assignments. To address this, Multi-View Network (MVNet) is introduced. This network learns to relate multi-view 2D predictions for single 3D points. The network is able to reduce classification errors in 2D with very little training data and outperforms other semi-supervised methods. By combining SNet and MVNet into Label Transfer Network (LTNet), the complete label transfer from 2D to 3D can be learned. LTNet works in both domains simultaneously and achieves a mIoU of 0.75 in 3D, which outperforms all previous models.

Moreover it is shown that it is possible to handle dynamic occlusions and self-occlusions in 3D through a self-supervised manner, i.e. without ground truth. Dynamic occlusions occur when moving objects appear in one domain but not in the other leading to incorrect object assignments. Here a Conditional Generative Adversarial Network (CGAN) is introduced that learns to map from 3D point clouds to 2D photorealistic images. Since the synthesized images and the point clouds match very well, this approach leads to much better results when mapping image labels belonging to dynamic classes such as cars to 3D point clouds. For self occlusions a GAN is introduced that learns to complete a range of 3D objects from incomplete observations only. The results show that the GAN performs almost similar to semi-supervised or fully-supervised methods, which helps in identifying occupied regions in 3D and could potentially lead to fewer errors in the label transfer process.


**Keywords: Deep Learning, Label Transfer, Semantic Segmentation**

## Kurzfassung

Die Semantische Segmentierung ist ein wissenschaftliches Teilgebiet der Computer Vision. Mit ihr sollen Maschinen das Verständnis von einer Umgebung erlangen, das ähnlich zur visuellen Wahrnehmung des Menschen ist. Eines ihrer Einsatzbereiche ist die autonome Mobilität. Dabei werden z.B. Autos mit verschiedenen Sensoren, wie Kameras und 3D-Laserscannern, ausgestattet, um ein vollständiges Verständnis von der Umgebung, in der sie fahren, zu erlangen. In den letzten Jahren wurde das Thema von tiefen neuronalen Netzwerken dominiert, die große Mengen an Trainingsdaten benötigen. Die Erstellung dieser Datensätze ist sehr zeit- und kostenaufwendig. Die Datensätze können zudem nur für einen bestimmten Sensortypen angewendet werden. Die vorliegende Arbeit behandelt dieses Problem. Es soll gezeigt werden, dass das Wissen aus öffentlich verfügbaren Bilddatensätzen wiederverwendet werden kann, sodass der Aufwand für das Annotieren (Labeln) von 3D-Punktwolken minimiert wird. Dafür werden die Label aus klassifizierten Bildern in 3D-Punktwolken übertragen. Um die Unterschiede zwischen den Sensormodalitäten zu überbrücken, wird der geometrische Zusammenhang der Sensoren in einem vollständig kalibrierten System verwendet. Es gibt unterschiedliche Fehlerquellen, die bewirken, dass die einfache Übertragung der Label zu einer erheblichen Menge an falschen Klassenzuordnungen in den 3D-Punktwolken führen. Im Rahmen dieser Arbeit werden die verschiedenen Ursachen und Lösungsmöglichkeiten aufgezeigt, um die Übertragung der Label zu verbessern.

Als erstes wird Scanstrip Network (SNet) präsentiert. Das Netzwerk lernt, falsche Klassenzuordnungen in 3D-Punktwolken zu korrigieren und berücksichtigt dabei implizit verschiedene Fehlerquellen. Hierbei wird es mittels Überwachten Lernens und anhand von wenigen Daten trainiert. Das einfache aber effektive Netzwerkdesign erreicht einen mIoU von 0.67 im Gegensatz zum Ausgangswert von 0.48 und übertrifft damit ähnliche und sogar modernste Netzwerke. Diese Ergebnisse werden weiter verbessert, indem SNet auf eine halb-überwachte Weise trainiert wird. Hierfür werden große Mengen automatisch generierter Labels für das Vortraining verwendet, wodurch das Netzwerk einen mIoU von 0.71 erreicht.

Ein Problem zu Beginn der Label-Übertragung sind Klassifikationsfehler in Bildern und falsche Zuordnungen von 2D-Pixeln zu 3D-Punkten. Um dies zu berücksichtigen, wird das Multi-View Network (MVNet) eingeführt. Dieses Netzwerk lernt, Vorhersagen von 2D-Mehrfachansichten für einzelne 3D-Punkte in Beziehung zueinander setzen. Das Netzwerk ist in der Lage, Klassifizierungsfehler in 2D mit nur sehr wenigen Trainingsdaten zu reduzieren und übertrifft andere halbüberwachte Methoden. Durch das Kombinieren von SNet und MVNet zu Label Transfer Network (LTNet) kann die komplette Label-Übertragung von 2D zu 3D gelernt werden. LTNet arbeitet in beiden Domänen gleichzeitig und erreicht ein mIoU von 0.75 in 3D, was alle bisherigen Modelle übertrifft.

Darüber hinaus wird in der vorliegenden Arbeit gezeigt, dass es möglich ist, dynamische Verdeckungen und Selbstverdeckungen in 3D auf selbstüberwachte Weise und somit ohne Label zu behandeln. Dynamische Verdeckungen treten auf, wenn Objekte in einer Domäne erscheinen, aber nicht in der anderen. Dies führt zu falschen Objekt-Zuordnungen. Hier wird ein Conditional Generative Adversarial Network (CGAN) eingeführt, das lernt, fotorealistische Bilder aus 3D-Punktwolken zu erzeugen. Da die synthetisierten Bilder und die Punktwolken übereinstimmen, führt dieser Ansatz zu wesentlich besseren Ergebnissen bei der Zuweisung von Labels im Falle von dynamischen Verdeckungen. Für Selbstverdeckungen wird ein GAN eingeführt, das selbständig lernt, 3D-Objekte zu vervollständigen. Die Ergebnisse zeigen, dass das GAN ähnliche Leistungen erbringt wie halb- oder vollüberwachte Methoden. Dies hilft bei der Identifizierung besetzter Regionen in 3D, was möglicherweise zu weniger Fehlern im Übertragungsprozess führen könnte.

**Schlagworte: Tiefes Lernen, Label Transfer, semantische Segmentierung**

# Contents


**1 Introduction** **1**

**2 Theoretical Background** **7**

2.1 Cameras and Laserscanning . . . . . 7

2.1.1 Cameras . . . . . 7

2.1.2 Laserscanning . . . . . 10

2.2 Machine Learning Fundamentals . . . . . 19

2.2.1 Types of Learning . . . . . 19

2.2.2 Supervised Learning - Illustrated by Decision Trees . . . . . 20

2.2.3 Boosting . . . . . 23

2.3 Deep Learning . . . . . 25

2.3.1 Basics . . . . . 25

2.3.2 Self-Attention . . . . . 30

2.3.3 Generative Adversarial Networks . . . . . 32

**3 Related Work** **35**

3.1 Classification and Semantic Segmentation (2D) . . . . . 35

3.2 Semantic Segmentation (3D) . . . . . 39

3.3 Semi-Supervised Learning . . . . . 42

3.4 Conditional Generative Adversarial Networks . . . . . 45

3.5 Multi-View Fusion, Prediction and Labeling . . . . . 45

3.6 Shape Completion . . . . . 47

**4 Multi-View Label Transfer and Correction** **49**

4.1 2D to 3D Label Transfer . . . . . 50

4.1.1 Regular and Self-Occlusions . . . . . 50

4.1.2 Dynamic Occlusions . . . . . 51

4.1.3 Naive Label Transfer and Label Policy-Based Noise . . . . . 53

4.2 Label Noise Correction . . . . . 54

4.2.1 Scanstrip-Based Noise Correction . . . . . 54

4.2.2 Semi-Supervised Scanstrip-Based Noise Correction . . . . . 56

4.2.3 Conclusion . . . . . 58

4.3 Multi-View Outlier Correction and Label Transfer . . . . . 59

4.3.1 Multi-View Network . . . . . 59

4.3.2 Label Transfer Network . . . . . 63

4.3.3 Conclusion . . . . . 64

**5 Self-Supervised Point Cloud Rendering and Completion 65**
5.1 Photo-Realistic Point Cloud Rendering . . . 66
5.1.1 Network Architecture . . . 66
5.1.2 Loss Function . . . 68
5.1.3 Image Stitching . . . 69
5.2 Self-Supervised Shape Completion . . . 70
5.2.1 Subregion-Based GAN model . . . 70
5.2.2 Loss Function . . . 72
5.2.3 Network Architecture . . . 74

**6 Preparation of MMS data 77**
6.1 Preprocessing of the Mobile Mapping Dataset . . . 77
6.1.1 Semantic Segmentation of the MMS-Dataset . . . 78
6.1.2 Human annotated MMS-Dataset . . . 79
6.2 Massively Parallel Point Cloud Rendering Using Hadoop . . . 81
6.3 Datasets of Self-Occluded Objects . . . 83
6.3.1 Real Dataset . . . 84
6.3.2 Synthetic Datasets . . . 85

**7 Experiments and Results for Multi-View Label Transfer 87**
7.1 Introduction . . . 87
7.2 Baseline . . . 87
7.3 Training, Validation and Test Set . . . 92
7.4 Scanstrip-Based Correction . . . 93
7.4.1 Point-Wise Correction . . . 93
7.4.2 Supervised Scanstrip-Based Correction . . . 95
7.4.3 Semi-Supervised Scanstrip-Based Correction . . . 98
7.4.4 Qualitative Evaluation . . . 99
7.4.5 Conclusion and Discussion . . . 104
7.5 Multi-View Error Correction . . . 105
7.5.1 Baseline . . . 105
7.5.2 Training, Validation and Test Sets . . . 106
7.5.3 Training Procedure . . . 108
7.5.4 Ablation Studies and Results . . . 109
7.5.5 Qualitative Evaluation . . . 113
7.5.6 Retraining Semantic Segmentation Networks . . . 114
7.5.7 Results of the Retraining Process . . . 116
7.5.8 Conclusion and Discussion . . . 118
7.6 Multi-View Label Transfer Learning . . . 120
7.6.1 Training Procedure . . . 120
7.6.2 Ablation Studies and Results . . . 121
7.6.3 Qualitative Evaluation . . . 124

7.6.4 Conclusion and Discussion . . . 124
7.7 Summary and Conclusion . . . 126

**8 Experiments and Results for Self-Supervised Completion 127**
8.1 Photorealistic Point Cloud Rendering . . . 127
8.1.1 Training Procedure . . . 127
8.1.2 Quantitative Evaluation . . . 128
8.1.3 Qualitative Evaluation . . . 130
8.1.4 Multi-View Error Correction in GAN Images . . . 134
8.1.5 Conclusion and Discussion . . . 139
8.2 Self-Supervised Shape Completion . . . 141
8.2.1 Training Procedure . . . 141
8.2.2 Quantitative Evaluation . . . 141
8.2.3 Qualitative Evaluation . . . 143
8.2.4 Conclusion and Discussion . . . 145

**9 Conclusion and Discussion 147**
9.1 Summary and Discussion . . . 147
9.1.1 Scanstrip-Based Label Error Correction . . . 147
9.1.2 End-To-End Multi-View Label Transfer . . . 148
9.1.3 Self-Supervised Completion . . . 149
9.1.4 Conclusion . . . 150
9.2 Outlook . . . 150

**List of Figures 153**

**List of Tables 161**

**Bibliography 163**

**Resume 176**

**Acknowledgements 177**

# 1 Introduction

## Motivation and Research Goals

For tasks such as mobile mapping or autonomous driving, cars are equipped with various sensors such as Global Navigation Satellite Systems (GNSSs), Inertial Measurement Units (IMUs), cameras and laser scanners. These sensors collect data that can be used for everything from mapping or localization to completely replacing a human driver. In some cases, the sensor data is preprocessed and then interpreted by machine learning algorithms such as Artificial Neural Networks (ANNs) to semantically understand the environment. A very common problem is that training such an algorithm is very expensive due to human supervision. Often a company has to be hired to create enough annotated data for training purposes. This increases costs and effort and can further complicate a project. Furthermore, these annotations can quickly become outdated or obsolete due to the introduction of newer sensor models or changing system requirements. To avoid the high costs of annotating data, much effort is put into reusing information or adapting already pretrained ANNs to new problems. In computer vision, this is often accomplished by training an Deep Convolutional Neural Network (DCNN) on publicly accessible datasets and then fine-tuning it to solve a similar task on the target dataset. It is desirable to use few or even no annotations in the target dataset, which is called semi-supervised or even unsupervised Domain Adaptation.

The goal of this work is to introduce **a framework for minimizing the need for annotations** for 3D point clouds and also 2D image data. The general idea is to take the information collected in publicly available (annotated) 2D image datasets and learn how to map the annotations to an unlabelled 3D dataset. A prerequisite for this framework is that the camera and laser scanner are fully calibrated, which is almost always the case in this setting. This means that the relative orientation between all sensors as well as the intrinsic sensor parameters are known and all recorded data is time-stamped. To illustrate the core idea of this framework Figure 1.1, shows three images of the same scene taken at different times from a moving vehicle. The red star should resemble a single 3D point that is detected by a laser scanner and projected into all images using the known calibration of the sensors, so that it marks the same location in all three images. The line below the images shows a possible prediction for each pixel located on the red star made by a DCNN pretrained on a publicly available dataset. In the first and third images, the DCNN correctly identifies the class *wall* at the marked pixels. However the pixel in the middle image was incorrectly labelled as *car*. With this information two problems can be addressed: (1) The incorrect prediction *car* can be corrected by assigning it the majority class *wall*. (2) All three predictions belong to the same point in 3D, so that a label can be assigned from these three predictions, i.e. the majority vote *wall*. This label transfer preserves knowledge from the 2D domain by labeling 3D point clouds with 2D image predictions, thus reducing or even eliminating expensive labeling costs of 3D point clouds.

However in the above example some simplifications and assumptions were made. In reality the naive label transfer will assign a lot of wrong labels to each 3D point (label noise) and can make the predictions in 2D even worse if the majority vote is assigned to each of them. A correct label is assigned to a 3D point if the majority among all predictions belongs to the actual class of the 3D point. However there are many reasons why this is often not the case. **First** the initial predictions of the DCNN can be wrong or too bad, which occurs when either the model predictions are too poor or it does not generalize well enough to the new camera sensor type or environment, a problem

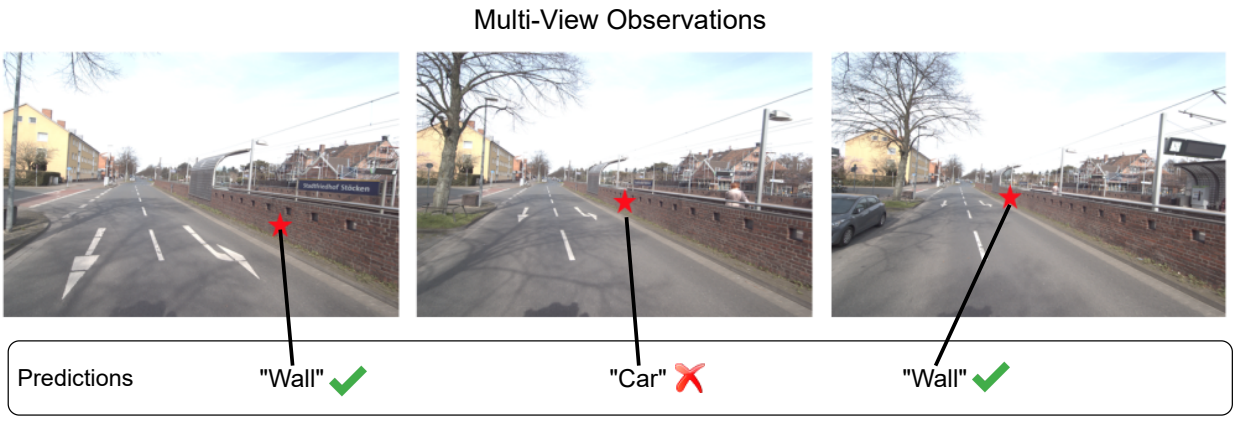



*Figure 1.1: An example showing three different images of the same scene. The red star marks the same spot in these images for which a prediction was made by 2D DCNN. Although all three pixels are in different positions in different images, they belong to the same object (wall). By linking the three predictions, the wrong car prediction can be detected and corrected to wall.*

often referred to as domain gap. This means that if the domain gap is too large, wrong predictions are no longer outliers and cannot be compensated by majority voting. **Second**, the assignment between 3D points and 2D pixels in the images can be wrong, which will group different objects in 2D together. This problem is even more severe because even if an object is correctly identified in the images, an incorrect label would be assigned to a 3D point and the 2D pixels if the pixels are incorrectly linked. The reason for incorrect linkage can be **calibration and occlusion errors**. As the calibration error increases, the 3D point is projected to the wrong location in the respective image. Even with a small offset, this error has an effect on small or thin objects such as poles, which are then incorrectly assigned to surrounding objects. Occlusion errors on the other hand, are divided into regular, self-occlusions and dynamic occlusions. Regular and self-occlusions occur when a 3D object is either measured too sparsely (regular occlusion) or parts of the object are not measured because the laser beams are blocked by the object itself (self-occlusion). In both cases this leads to problems when projecting the corresponding 3D points into an image, because occluded points in 3D that are not visible in the camera can still appear in the image. The reason for this is that the beam between the 3D point and the camera center is not blocked by any other point due to sparsity. This will lead to a wrong assignment in multi-view images and introduces label-noise. On the other hand dynamic occlusions occur because camera and laser beams often do not match. This means that moving objects can appear in an image but not in the point cloud or vice versa or the objects can appear at completely different locations. This often leads to confusion between static and dynamic object classes, which is especially common in urban environments. Finally incompatible labeling policies between the two domains can lead to label noise. In the case of the Cityscapes dataset made by Cordts et al. (2016) tree canopies are labelled as *vegetation*, including partially hidden background pixels belonging to different objects behind the tree. This leads to incorrect mapping of *vegetation* labels to 3D points located behind the tree such as facades and buildings.

In order to successfully use label transfer to minimise annotation costs, the previously described problems must be taken into account. Possible solutions should avoid requiring high volumes of ground truth labels, as this would render the label transfer useless. This is why the following research hypotheses are investigated in this thesis:

**Research Hypothesis 1** *Wrong classification, domain gaps, calibration errors, incompatible label policy and self-, regular and dynamic occlusions are causes of label noise in $2D \rightarrow 3D$ label transfer.*

This hypothesis forms the baseline for this work. The use of naive label transfer is based on the work of Peters and Brenner (2019). Here the label transfer is done without using learning based algorithms, by mapping 2D pixel predictions to 3D point clouds with majority voting which leads to a significant increase in label noise in 3D. To show this, two human annotated reference sets are introduced in this thesis. One contains 23 2D images and the other contains 14 different 3D scenes. By comparing the predicted labels in 2D and the transferred labels in 3D with the respective reference sets it is possible to measure the amount of label noise. Furthermore, by measuring the label confusion (in 2D) before and after the label transfer (in 3D), many of these errors and their impact can be illustrated. Finally, features are introduced to detect certain causes of label noise.

**Research Hypothesis 2** *Label noise caused by geometrically transferred labels can be corrected in 3D using a small reference set.*

This hypothesis states that it is possible to implement a late correction step after labels have already been assigned to 3D points. For this purpose, a 2.5D neural network is introduced in this thesis called Scanstrip Network (SNet) that is capable of correcting incorrectly assigned labels through supervised learning. The network is roughly based on the architecture by Ronneberger et al. (2015). Furthermore, it is shown that and to what extent the introduced features help to detect different types of errors in order to correct them. Finally, Scanstrip Network is compared to various state-of-the-art methods to show its superiority in correcting label noise.

**Research Hypothesis 3** *Naively transferred labels are useful as a supervision signal, even if they contain label noise. They can be used to learn initial representations resulting in better classification models compared to training supervised models with random weight initialization.*

This is an extension of the previous hypothesis. Instead of training Scanstrip Network in a supervised manner, a similar network will be trained in semi-supervised fashion in order to create a classifier that is better at label noise correction. For this purpose, a two step approach will be introduced: First the network is trained on all data learning to classify 3D points by using the transferred labels as pseudo-labels. In the second step, parts of the network will be frozen and the rest will be fine-tuned to the reference set to learn label noise correction. It will be shown that this approach outperforms the previous approaches.

**Research Hypothesis 4** *First: Multi-view images can help to increase the classification performance in 2D using a suitable multi-view network architecture. Second: The use of Multi-View Network (MVNet) for knowledge distillation, i.e., fine-tuning a normal 2D DCNN on pseudo labels created by MVNet in the target domain, outperforms standard supervised training of a DCNN on the targets.*

The hypothesis is based on the work of Peters et al. (2020). They showed that errors in 2D semantic segmentation can be addressed by combining multiple predictions from the same network but from multiple views. Based on this, Multi-View Network is introduced, which is a significant improvement of the network of Peters et al. (2020). Similar to the example in Fig. 1.1, the Multi-View Network receives a list of multi-view images and corresponding class predictions made by a pretrained network. All multi-view images are cropped so that the central pixel belongs to the same object in 3D (red stars). The output of the network is a class prediction for each central pixel of each image (red stars). It will be shown that the MVNet achieves a higher mIoU than a similar network that only has access to single images. Moreover, the predictions made by the MVNet can be used to fine-tune a pretrained DCNN on the target domain, which outperforms fine-tuning of the same DCNN directly on the targets.

**Research Hypothesis 5** *Naive label transfer can be replaced by learning to transfer labels from 2D to 3D end-to-end, resulting in much less label noise while requiring only a few reference labels.*

So far all methods have dealt with label noise after the labels have been aggregated in 3D. Here the Label Transfer Network (LTNet) is introduced, which is able to transfer the initial predictions of the DCNN from multi-view images to the 3D point cloud by using a combination of the Multi-View and Scanstrip Networks. More specifically, the network receives a list of multi-view images with the initial predictions of a pretrained DCNN, along with features extracted by the Scanstrip Network in 3D. The output of the network is a class prediction for the corresponding 3D point. This network is shown to outperform all approaches that attempt to correct label noise in 3D after the transfer.

**Research Hypothesis 6** *Synthesized multi-view images created from point cloud data can replace real camera images for label transfer and also mitigate errors by dynamic occlusions.*

This hypothesis is based on the work by Peters and Brenner (2020). Here, a Conditional Generative Adversarial Network (CGAN) is used to generate photorealistic images from point cloud data. The input of the CGAN is a projected point cloud image and the output is a generated RGB camera image. The network is evaluated in various ablation studies for its performance in generating realistic looking images and its generalization ability. Additionally, the CGAN will be fully integrated into the label transfer process: Once trained, the CGAN can replace the camera in a fully calibrated system. Moreover the synthesized images are realistic enough to be interpreted by a pretrained DCNN so that they can be used for label transfer. As the CGAN makes its predictions based on point cloud images, there is less dynamic occlusion, because dynamic objects such as cars appear in the same location in both domains. It is shown that this approach significantly reduces the label noise for dynamic objects in naive label transfer.

**Research Hypothesis 7** *It is possible to learn to complete self-occluded 3D objects in a self-supervised manner from incomplete data.*

Since self-occlusions are a cause of label noise in the label transfer process, it is shown that it is possible to learn to complete self-occluded objects without supervision. The presented CGAN is trained on a dataset of roughly aligned objects of one class, all suffering from self-occlusion. The input to the generator will be an incomplete 3D object surface in a voxelized representation and the output will be the corresponding complete object. To demonstrate that the CGAN can operate on real data in an unsupervised manner, the naive label transfer method will be used to automatically extract nearly 9000 roughly aligned car-scans. It is shown that the CGAN is able to complete these scans and furthermore is able to generalize to car scans with different characteristics from the KITTI dataset by Geiger et al. (2013). To demonstrate that the CGAN works on other object types such as planes or even chairs, several ablation studies are performed on synthetic shapes obtained from the Shapenet and Modelnet databases by Chang et al. (2015) and Wu et al. (2015).

### Outline

The thesis will be organized as follows:

- In Chapter 2 an introduction to the theoretical background related to this thesis is given. First, the sensors used (LiDAR and camera) are discussed. Subsequently, the basics of machine learning as well as deep learning are covered here.
- Chapter 3 provides a detailed introduction to related work including state-of-the-art approaches. First, semantic segmentation in 2D and 3D is discussed. Then, semi-supervised and self-supervised learning, conditional adversarial networks, multi-view-based models, and shape completion are covered.
- Chapter 4 and 5 discusses the general methodology. These chapters form two blocks. The first one covers fully and semi-supervised methods for label noise correction. The second one covers self-supervised methods.
- Chapter 6 describes the data used and the reference set. It is shown in detail how the datasets for the experiments were created.
- Chapters 7 and 8 show the results for the methods, described in Chapters 4 and 5, forming two blocks that validate the proposed research hypotheses.
- Chapter 9 concludes the entire thesis and summarizes the findings. Finally, further research is discussed.

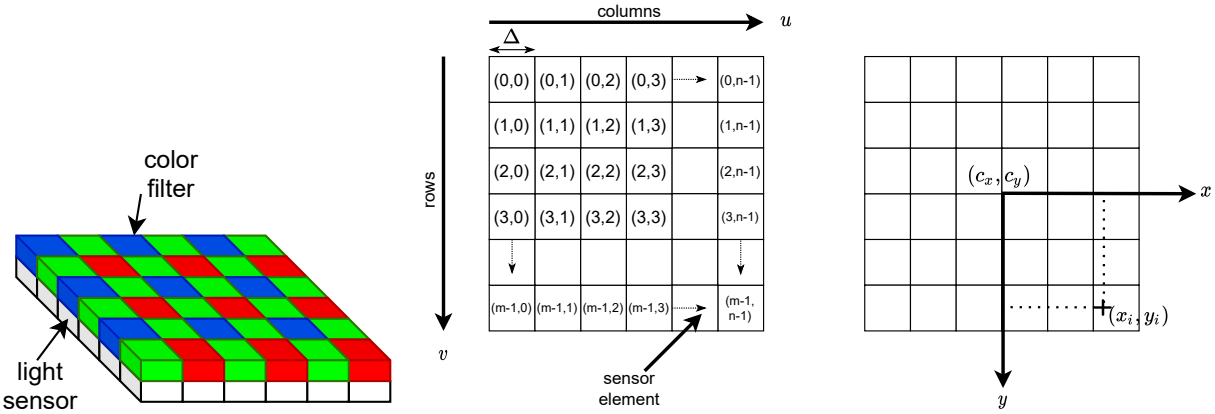
# 2 Theoretical Background

## 2.1 Cameras and Laserscanning

### 2.1.1 Cameras

Cameras are of particular interest for autonomous driving because they offer high resolution data at relatively low cost. They can be mounted on multiple sides of an autonomous car, which can be combined to form a wide field of view. These sensors help the car navigate through urban environments and avoid accidents. Popular use cases include object recognition, semantic segmentation, 3D reconstruction from multiple images, mapping, and localization. However, cameras also have their limitations, as they do not provide distance information that must be reconstructed using, for example, stereo matching or structure from motion. They are also passive sensors, which makes it difficult to detect objects in poor visibility conditions such as night, rain, or fog.

#### Fundamentals and Perspective Projection




*(a)* Bayer pattern (red green and blue) on an image sensor (grey)

*(b)* The pixel coordinate system with pixel size Δ

*(c)* The image coordinate system

*Figure 2.1: A camera sensor measures the RGB values using a Bayer pattern (left). The values are stored in the pixel coordinate system (middle). The projected image on the sensor is given in the image coordinate system (right)*

Cameras project 3D objects through lenses onto light-sensitive sensors, from which a 2D image is derived. Common sensors are Charge Coupled Device (CCD) or Complementary MetalOxide Semiconductor (CMOS) sensors that capture light on a rectangular sensor grid, where each cell is a photoelectric cell. To derive color information, the light is filtered by filters arranged in a Bayer pattern (Förstner and Wrobel, 2016, p. 444) (Fig. 2.1a), where each filter only allows a specific color to pass through to a single light sensor. Typically, this filter is arranged to measure the red, green and blue values from adjacent sensors. This Bayer pattern image can be converted into an RGB image, which is called demosaicking. This is usually done by interpolating the neighboring measured red, green and blue values for each pixel in the RGB image.

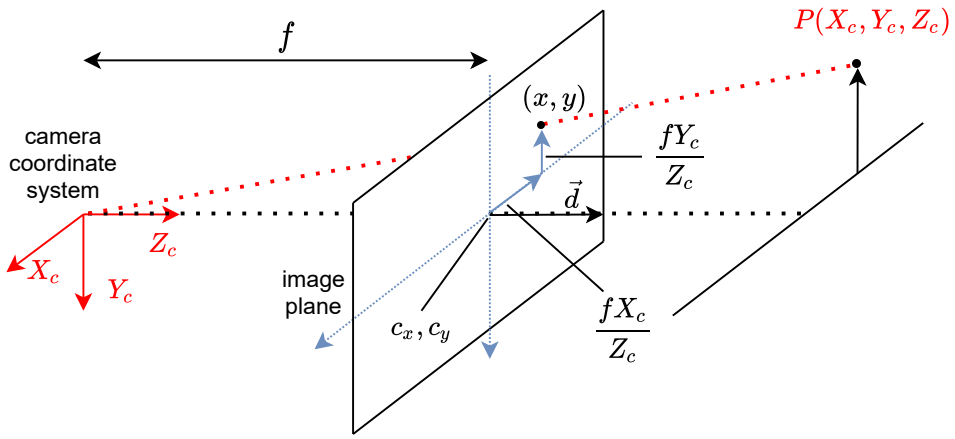



*Figure 2.2: Schematic relationship between a point in camera coordinates and the image plane*

The very basic principle of projecting objects to a plane has been known for a few centuries. The pinhole camera captures the reflected light of the environment on a plane through a very small opening. The image of the object is created by the intersection of the light rays with the image plane. Such a mapping from three dimensions onto two is called perspective projection as described in (Förstner and Wrobel, 2016, p. 253-257). This simple device already defines some aspects of digital cameras in photogrammetry and computer vision. The pinhole defines the camera center and the distance between projection center and image plane is the focal length $f$, also referred to as camera constant (Fig. 2.2). As the actual image on the surface is horizontally flipped most visualizations are showing the image plane in front of the projection center.

The values on a digital sensor are measured in the **pixel coordinate system** (Fig. 2.1b), the axes of which are given in column $u$ and row $v$. The origin of the pixel coordinate system is in the middle of the upper left pixel, see Figure 2.1b. However the y-axis points downwards, parallel to the image row direction.

The **image coordinate system** is shown in Fig. 2.1c. Unlike the pixel coordinate system, the image coordinate system can be specified in a metric system and has its origin in the center of the image. The axes can be defined differently, here the x-axis is parallel to the image columns and points in the same direction as the x-axis in the pixel coordinate system. Similarly, the y-axis is parallel to the rows and points also in the same direction. There are also other definitions where one or more coordinate axes are flipped, e.g. (Förstner and Wrobel, 2016, p. 471). The optical axis $O$ is perpendicular to the image plane. The distance between the lens and the sensor array is $f$. The intersection of the optical axis with the image plane is called principal point $c_x$,$c_y$, which is given in the image coordinate system, s. Fig. 2.1c. The principal point lies approximately in the center of the image. Additionally there might be a skewness coefficient $s$, which might be non-zero if the pixel axes are not perpendicular.

The following equation shows how to transform a point $[X_c,Y_c,Z_c]$ from the camera coordinate system (defined below) into the pixel coordinate system.

$$\begin{aligned} u_i &= (f\frac{X_c}{Z_c} + c_x) \\ v_i &= (f\frac{Y_c}{Z_c} + c_y), \end{aligned} \tag{2.1}$$

where $u_i$ and $v_i$ are the column and row index of the pixel in the 2D array, respectively.

However, cameras in the real world use optical lenses to collect and focus light, which can cause distortion. The most important type is radial distortion, in which the incoming light rays are bent toward or away from the center. The effect is radially symmetric and is often modelled as a polynomial of the distance to the optical centre (Förstner and Wrobel, 2016, p. 508-510).

All these parameters are called the interior orientation (also intrinsic parameters) of the camera and are usually estimated during a camera calibration. They model the geometry of the camera in order to infer the direction of the projection beam to an object point from an image point and the external orientation. Besides the interior orientation with non linear distortions six additional parameters are needed for the exterior orientation in order to approximate a real camera (Förstner and Wrobel, 2016, p. 464).

The objects are defined in the world coordinate frame which is in 3D ($X_w$,$Y_w$,$Z_w$). The camera pose in the world coordinate frame is given by the camera position $t_x$,$t_y$,$t_z$ and orientation $\phi$,$\theta$,$\psi$. To relate the 3D world coordinates to the 2D image plane, i.e. to map from world coordinates to the camera coordinate system, a homogeneous transformation can be used as a mechanism to form a compound transformation. Homogeneous coordinates extend the dimensionality of the domain space so that classical transformations can be expressed linearly (Förstner and Wrobel, 2016, p. 250f.). Vectors in homogeneous space add a new parameter $\omega$ that defines the scale along the new axis. For example, a point in homogeneous coordinates is $p = [u,v,\omega]^T$. Let $R(\phi,\theta,\psi) \in \mathbb{R}^{3\times3}$ be a rotation matrix that gives the orientation of the camera and $t = [t_x,t_y,t_z]^T$ be a translation vector that defines the position of the camera in the world coordinate frame. The transformation from the world coordinate frame $w$ into the camera coordinate frame $c$ can be expressed in a single matrix $T \in \mathbb{R}^{3\times4}$ of the following form

$$T = \begin{bmatrix} R & t \end{bmatrix} \tag{2.2}$$

In the same way the intrinsic parameters can be used to formulate a matrix that projects a point from the camera frame into the image frame with $K \in \mathbb{R}^{3\times3}$. As this matrix uses the intrinsic parameters it is also referred to as calibration matrix.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

In order to project a 3D point $p_w = [X_w,Y_w,Z_w]^T$ from the world coordinate frame into the image plane the exterior and interior orientation is used together.

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = K \cdot T \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.4}$$

Since the matrices are expressed in homogeneous coordinates the point $p_w$ needs to be expanded by one dimension before it can be projected, Equation 2.4. After all transformations have been applied the point $[u',v',w']^T$ needs to be normalized to yield an Euclidean representation:

$$u = \frac{u'}{v'},\ v = \frac{v'}{w'}, \tag{2.5}$$

where $(u,v)$ are the image coordinates.

With a virtual camera, 3D points behind the camera could be projected onto the camera plane. To prevent this, a visibility check can be performed. For this purpose, the viewing direction vector $\vec{d}$ is used, which is perpendicular to the image plane and points in the viewing direction, s. Fig 2.2. Since the dot product between two vectors is zero when both vectors are perpendicular to each other and increases the further both vectors point in the same direction, the view frustum frustum($\vec{d}$,$p_w$) between $\vec{d}$ and the 3D point $p_w$ should be greater than zero:

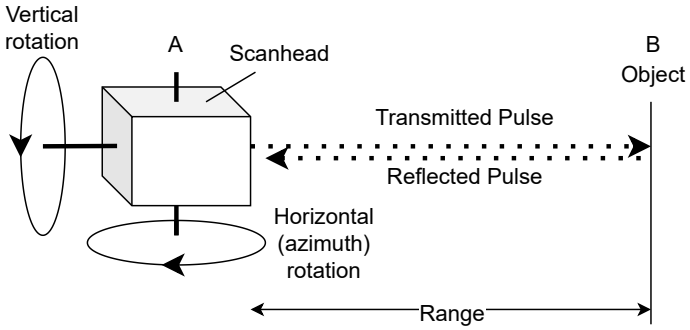
$$\text{frustum}(\vec{d},p_w) = \begin{cases} True, & \text{if } \vec{d}p_w^T > 0 \\ False, & \text{otherwise} \end{cases} \tag{2.6}$$

If the function returns $True$, the 3D point lies within the camera's viewing frustum and can be projected onto the camera plane.

### 2.1.2 Laserscanning




*(a)* The RIEGL VMX-250 Mobile Mapping System equipped with two line scanners




*(b)* Schematic view of an time-of-flight laser scanner with two degrees of freedom

*Figure 2.3: Two images showing a pair of real laser scanners (left) and a schematic drawing of a time-of-flight (ToF) laser scanner (right)*

#### Introduction

A laser scanner or Light Detection And Ranging (LiDAR) sensor is an active sensor for measuring distances. Laser scanners are operated from airborne, terrestrial or mobile platforms (Vosselman and Maas, 2010). The laser scanner emits a beam of light that is reflected from the measured object surface. By measuring the time that elapses between the emission and the reception of the pulse, the distance between the laser scanner and the surface is calculated by multiplying the travel time by the speed of light. By combining the measured distance with the orientation of the scanner head, the exact 3D position of the measured point can be calculated in the scanner coordinate frame. The dimension of the measurement of the device, depends on the total number of degrees of freedom. In reality, the horizontal measurement is realized by a rotating mirror that reflects the laser beam and encodes the current angle; the vertical rotation is then realized by moving the entire scanner head, Fig. 2.3b. Laser scanners are usually equipped with infrared lasers with a typical wavelength between 800 and 1550 nm (Vosselman and Maas, 2010, p.25.). The measurement of distance or range is always based on the accurate measurement of time and is performed using one of the two main methods, which are based on time-of-flight or phase. In the following, the time-of-flight method is described, since it is also used in this work.

### Time-of-Flight-Based

time-of-flight (ToF) is a method for distance measurement. It involves the precise measurement of the travel time of a very short laser pulse. In this way, the instrument measures the exact time interval that elapses between the emission of the pulse at point A, the reflection at point B and the re-reception at point A, see Figure 2.3b. The slant distance or range $r$ is calculated using the known speed of light $c = 299792458\ \frac{\mathrm{m}}{\mathrm{s}}$ and the measured time interval $t$.

$$r = \frac{c \cdot t}{2} \tag{2.7}$$

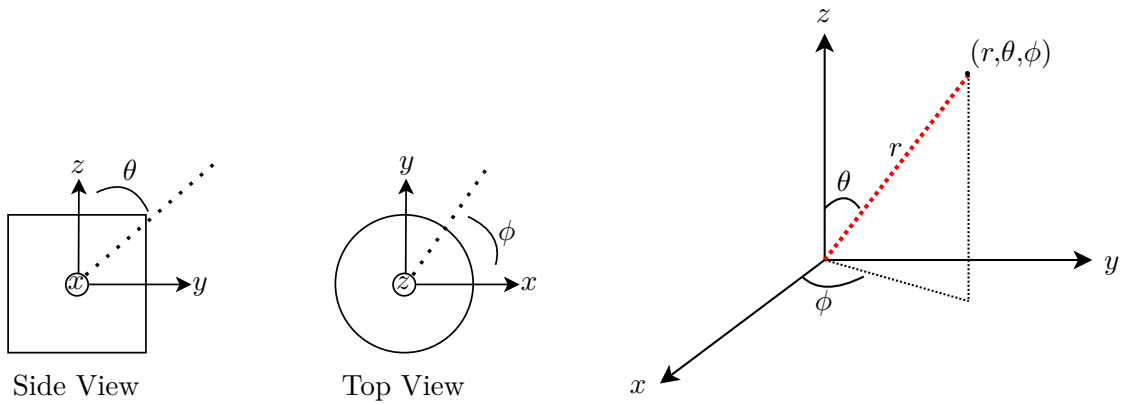
If the light travels through air then a small correction factor $\frac{1}{n}$ is multiplied to the velocity $c$. The value depends on air temperature, humidity and pressure (Vosselman and Maas, 2010, p.3). Since the speed of light is known very accurately, in practice the accuracy or resolution of the distance is determined by the accuracy of the time measurement. However, a pulse may return more than one echo due to location characteristics such as irregularly shaped ground or the fact that the laser spot increases in diameter with distance and hits different targets (Vosselman and Maas, 2010, p.3). This for example happens when airborne laser scanners capture a forest. They scan through the canopy of leaves and also measure ground points. According to Vosselman and Maas (2010, p.4) a typical laser pulse has a rise time $t_p = 5$ ns, which corresponds to a total length of 1.5 m. If a laser beam returns two echoes, they can only be distinguished if their distance is greater than half the pulse width. This means that with a pulse length of $t_p = 5$ ns objects can be detected as separated if their distance is greater than 0.75 m (Vosselman and Maas, 2010, p.5).

As described by Vosselman and Maas (2010, p.5), there are three types of pulse detection: (I) **Peak detection** sets the trigger pulse to the maximum amplitude of the echo. The ToF is measured between the highest transmitted and received laser amplitude. This method can be problematic if the reflected light provides more than one peak. (II) **Thresholding** sets the trigger when the rising edge exceeds a fixed threshold, which has the disadvantage that weak echoes are not detected by the laser scanner. (III) By **Constant Fraction Detection**, which sets a trigger when a preset fraction of the maximum amplitude is reached. This has the advantage of being more independent of the echo amplitude.

The amplitude of an echo is proportional to the reflectivity of the scanned surface. Weakly reflective targets provide a lower amplitude and highly reflective targets such as retroreflective road markings or traffic signs provide a higher amplitude. Due to this fact, very simple pulse detectors tend to calculate larger ranges for less reflective targets, which can lead to effects such as road markings floating above the ground. This effect must be corrected, especially for Type I and Type II pulse detectors; constant fraction detection is less affected by this effect (Vosselman and Maas, 2010, p.15). Apart from that, the reflectivity provides useful information, it can be used, for example, to visualize the measurements in order to distinguish different surfaces.

### Scanning and Projection Mechanisms

There are many different types of laser scanners, depending on the environment and specific requirements such as field of view, distance and density. For example, in urban environments, where surrounding traffic can get quite close to the scanner, there is certainly a different requirement for the type of scanner than for an airborne laser scanner, which has to cover the ground from a great height. According to Vosselman and Maas (2010), there are at least five different laser scanning mechanisms, which can be an oscillating mirror, a rotating polygon, a Palmer scanner, a fiber scanner or a flash LiDAR (Vosselman and Maas, 2010, p. 17). In all cases, they usually have different scanning patterns and different properties w.r.t. density or distribution. **Oscillating**




*(a)* Scanner head seen from side and top view

*(b)* The spherical coordinate system of a LiDAR with the ray shown in red

*Figure 2.4: The LiDAR measures the range, horizontal and angular rotation given directly in a spherical coordinate system*

**Mirrors** are perhaps the simplest form of scanning mechanism. Here the laser beam is shot onto a rotating mirror that reflects the emitted pulse and also captures the reflected pulse. Oscillating mirrors are commonly used in commercial airborne systems where the laser beam is swept across the trajectory, resulting in a unique sinusoidal zigzag pattern on the ground. **Rotating polygons** are very similar, here instead of a flat mirror the surface of the mirror is formed by a polygon, resulting in more uniform patterns on the ground where the lines are almost parallel. In the **Palmer scanner**, the mirror surface is mounted at an angle of less than 90° to the rotation axis. This results in elliptical patterns on the ground with a rather small field of view. The **fiber optic scanner** sequentially passes laser pulses to a linear array of optical fibers via a rotating mirror. Since the fibers are fixed, this mechanism is very stable and has a fixed scan angle. Finally the **flash LiDAR** projects laser beams from a fixed array of diodes. They capture the reflected light with another light sensor like CMOS (Vosselman and Maas, 2010, see p. 19), resulting in a 3D image without mechanical parts.

In all cases, LiDAR requires not only the distance information, but also the direction in which the laser is directed to calculate the 3D point. In terrestrial systems this is typically done by measuring the polar angles $\phi$ and $\theta$ of the scanner head (Vosselman and Maas, 2010, p.16 3.2).

Figure 2.4a illustrates the vertical and horizontal angles of the laser scanner. Technically, these are measured using wheel encoders. The angle measurements are typically evenly distributed across the LiDAR's field of view. Of course, the precision and angular resolution of the encoder contributes to the accuracy of the laser scanner. The spherical coordinate system of a LiDAR is shown in Figure 2.4b with a LiDAR with two degrees of freedom $\theta$ and $\phi$ and the measured distance $r$ in red. With these angles, the Cartesian point coordinates $p = [x,y,z]$ can be calculated as follows:

$$\begin{aligned} x &= r \cdot sin\ \theta \cdot cos\ \phi \\ y &= r \cdot sin\ \theta \cdot sin\ \phi \\ z &= r \cdot cos\ \theta \end{aligned} \tag{2.8}$$

There are also devices such as the RIEGL VQ-250 shown in Figure 2.3a that have only one degree of freedom. They measure only distance and horizontal rotation, which is why they are called line scanners. A line scanner that scans parallel to the ground plane only outputs a 2D map of the

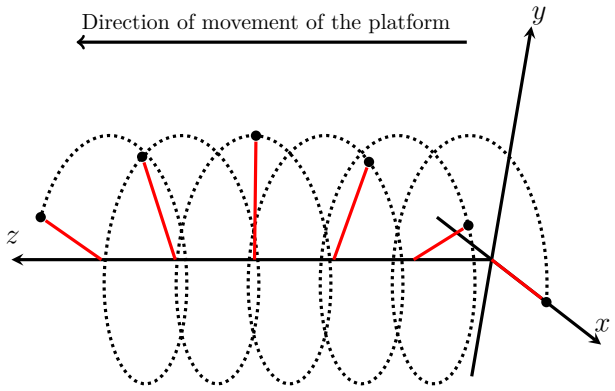



*Figure 2.5: Helical line scanner pattern (points) with scanner continuously rotating* 360° *along* $\phi$ *(Eq. 2.8) while moving in* $z$ *direction. The image corresponds to the coordinate system in Figure 2.4b with fixed* $\theta = 90°$*. Each red line shows a laser beam after the scanner has been rotated* 396° *and moved along the* $z$ *axis. In total, the scanner head is rotated 5.5 times in this plot.*

environment, which looks like a floor plan. However, if the line scanner is mounted on a rigid and calibrated system that provides access to a 3D position and heading in a global coordinate system, it is possible to calculate a 3D coordinate in the same system. This is usually realized via a combination of a GNSS receiver and an IMU.

### Mobile Mapping

Similar to airborne laser scanning, Mobile Mapping describes a measurement process performed on a moving vehicle. Typically, a measurement platform equipped with a variety of sensors such as GNSS, IMU, Distance Measuring Instruments (DMI), camera, and LiDAR is attached to the vehicle to survey its environment. Often, the combination of IMU, DMI, and GPS is used for a tightly coupled Kalman filter that estimates position, velocity, roll-, pitch-, and heading-angles of the sensor platform. Additionally, Simultaneous Localization and Mapping (SLAM) can be used to integrate LiDAR and camera observations into the filter steps (Puente et al., 2013). Other requirements include precise real-time synchronization and calibration between all sensors (Vosselman and Maas, 2010, p.301). The result of the mapping process are georeferenced 3D point clouds, often in a common global world system such as World Geodetic System 1984 (WGS 84) (Vosselman and Maas, 2010, p.294). In this context, mobile mapping is often associated with cars equipped with a Mobile Mapping System (MMS) such as RIEGL's VMX-250, but there are also a variety of complete systems for railroads (RIEGL VMX-RAIL), Unmanned Aerial Vehicles (UAVs) (YellowScan Fly & Drive or Li et al. (2017)), boats (Puente et al., 2013), human backpack systems (Leica Pegasus), or even indoor applications (Vosselman and Maas, 2010, p.301). Although image-based mobile mapping exists, as shown by Cavegn and Haala (2016), only LiDAR-based systems are discussed in this section.

According to Vosselman and Maas (2010, p.295ff.), Mobile Mapping is divided in "stop-and-go" and "on-the-fly" modes . The former describes many sequential static mapping processes using a vehicle-mounted mapping platform that is not moved during the mapping process. Ideally, each mapping step overlaps with the others so that the resulting point cloud can be registered using, e.g. the Iterative Closest Point (ICP). In "on-the-fly" mode, the vehicle continuously records data and does not have to stop. This process is much more efficient and captures larger areas in less time. As shown in Fig. 2.5, the line scanner is oriented with respect to the vehicle coordinate system so that it continuously scans inclined to the trajectory of the mapping platform. In this way, the scanner captures many consecutive 2D scan profiles that can be mapped into 3D using

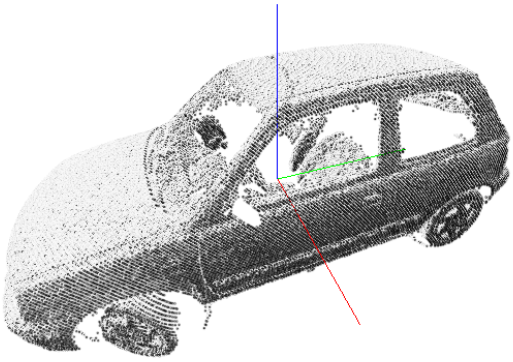


*Figure 2.6: 3D point cloud visualization of a car scanned with the RIEGL VQ-250 using the VMX-250 mobile mapping system.*

the estimated 3D pose of the mapping platform. The fixed inclination of the line scanner serves two main purposes. First it has the advantage that the scanner can reach very low objects like a road as well as high objects like facades, tree tops or even ceilings. Secondly, the scanner does not need to change the vertical orientation, which simplifies the whole system.

#### 2.1.2.1 3D Data Representations

There are a variety of different 3D data representations. The representation of 3D data is adapted to the requirements of the application. In this section, some standard representations and their applications are presented, which are necessary for the understanding of this work.

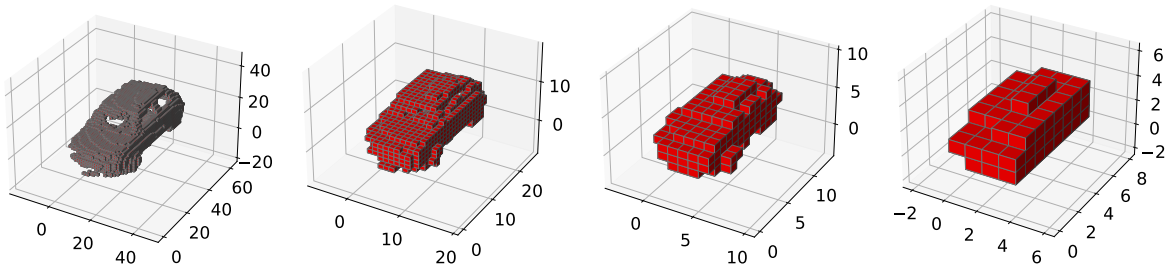
##### Point Clouds

An example of a 3D point cloud representation of a car acquired with the RIEGL VMX-250 MMS can be seen in Figure 2.6. Typical 3D sensors measure surfaces point by point in the sensor coordinate system. The 3D coordinates in the sensor coordinate system are calculated with the Equation 2.8. The relationship between the point in the sensor coordinate system $P_{laser}$ and the point in the global coordinate system $P_{obj}$ is described in Equation 2.9.

$$r \begin{pmatrix} cos(\phi) \\ sin(\theta) \\ 0 \end{pmatrix} = P_{laser} = R_{mount}^{T} \left[ R_{plat}^{T}(t) \left[ P_{obj} - P_{plat}(t) \right] - P_{mount} \right] \tag{2.9}$$

Here the trajectory of the MMS at time-step $t$ is given by $P_{plat}(t)$ and $R_{plat}(t)$. The fixed calibration for the mounting is given by the fixed transformation $R_{mount}$ and $P_{mount}$. To store these values, each 3D coordinate is appended to a list as a floating point number, supplemented with additional information such as RGB colors, reflectance, or labels. Thus, a point cloud can be stored in an unsorted vector $P^{n \times m}$, where each row consists of an observation of length $m$, e.g. $\{x_i,y_i,y_i,r_i,g_i,b_i\}$ with $i \in \{1,\ldots,n\}$. A drawback of this representation is that the entries of the vector are in no particular order, so finding a neighboring point requires $\mathcal{O}(n)$[1] operations, which is significantly

[1] $\mathcal{O}(\cdot)$ is used to described algorithms according to how their runtime or space requirements grow with the size of the input



*(a)* $r = 32 \times 62 \times 26$ $\rho = 6.3$ % $e = 6$ cm

*(b)* $r = 13 \times 25 \times 11$ $\rho = 17.2$ % $e = 15$ cm

*(c)* $r = 7 \times 13 \times 6$ $\rho = 31.7$ % $e = 30$ cm

*(d)* $r = 4 \times 8 \times 4$ $\rho = 53.1$ % $e = 50$ cm

*Figure 2.7: Different voxelizations of the point cloud of the car shown in Figure 2.6. The volume size is given by $r$, the density by $\rho$, and the edge length by $e$.*

more than the number of operations required in a grid-like structure with $\mathcal{O}(1)$. To improve this, a point cloud can be stored in a k-dimensional tree (kd-tree) introduced by Bentley (1975). This data structure stores k-dimensional points in a tree-like structure and allows more efficient nearest neighbor search with a time complexity of $\mathcal{O}(log\ n)$. The kd-tree has a space requirement of $\mathcal{O}(n)$, which is equivalent to one of a point cloud vector. However, depending on the algorithm used, the construction of the tree is more complex and has a worst-case complexity of $\mathcal{O}(kn\ log\ n)$ (Brown, 2015). The dimensionality $k$ of the tree is defined by the dimensionality of the data, i.e., in the case of a point cloud $k = 3$. In a kd-tree with $m$ levels, each level $i$ is assigned a particular dimension $d = i \mod k$. The tree is constructed by cycling through each axis $d$ and choosing a pivot value, for example the median. Points with a value higher than the median are then added to the nodes to the right of the pivot, and smaller values to the left. This is repeated until all points are added to the tree. To traverse a tree, the current node is compared to the query point $q$. If the query is smaller, the tree is traversed to the left, otherwise to the right. This is done separately for each dimension by traversing all k-dimensions.

Apart from the disadvantage that the search for neighbors is slower in point clouds than, for example, in grid-like structures, they have several advantages. First point clouds preserve the accuracy of the original data. Transformations can also be performed very quickly, as points can be multiplied by linear transformation matrices and also all points can be accessed independently, allowing programs to parallelize computing.

### Voxel Grids

A point cloud can be represented by a three-dimensional grid, similar to a two-dimensional image. Like pixels in a 2D image, the coordinates of a 3D voxel (volume element) are not explicitly encoded with their values, but are indexed by a data structure, see (Kaufman et al., 1993) for a detailed introduction. The major advantage of using voxel grids is that searching for neighboring points takes only $\mathcal{O}(1)$ operations. Voxel grids are also useful for estimating occlusions by approximating surfaces with box-shaped points. To voxelize a 3D point $p$ the following function can be used:

$$\text{Voxelize}(p) = \left\lfloor \frac{p}{e} \right\rfloor, \tag{2.10}$$

where $e$ defines the edge length of the voxel in meters. The result of this function is a list of integer 3D points, which can then be sorted into a 3D data structure with binary values 0,1, where 1 represents the presence of a point and 0 the absence (Kaufman et al., 1993). An example result of

the voxelization process is shown in Figure 2.7. Here, the scanned car from Figure 2.6 was voxelized with different edge lengths. The volume size $r$ was always chosen to closely fit the voxelized point cloud. By comparing these results, it can be seen that voxel grids have two major drawbacks. First, they have quantization errors that are larger for smaller voxel grids. This becomes clear when comparing 2.7a, where the car is still easily recognizable, to 2.7d, where the car can no longer be recognized. Second, they do not take advantage of sparsity, which means that memory requirements increase cubically with grid volume size. To reduce memory requirements, octrees can be used instead of regular grids as presented by Meagher (1982) or Laine and Karras (2010). In octrees, each node has eight children, hence its name. The root node represents all the data and each child node corresponds to an octant. This is repeated for each node until no further subdivision is possible or necessary. There are many advantages of using octrees, other than more efficient data representation. Each level of the octant system represents a different level of detail of the point cloud. This feature can be used for faster ray tracing, as the search in the search space for possible occupied grids is reduced, since child nodes that do not contain points are empty and do not need to be traversed.

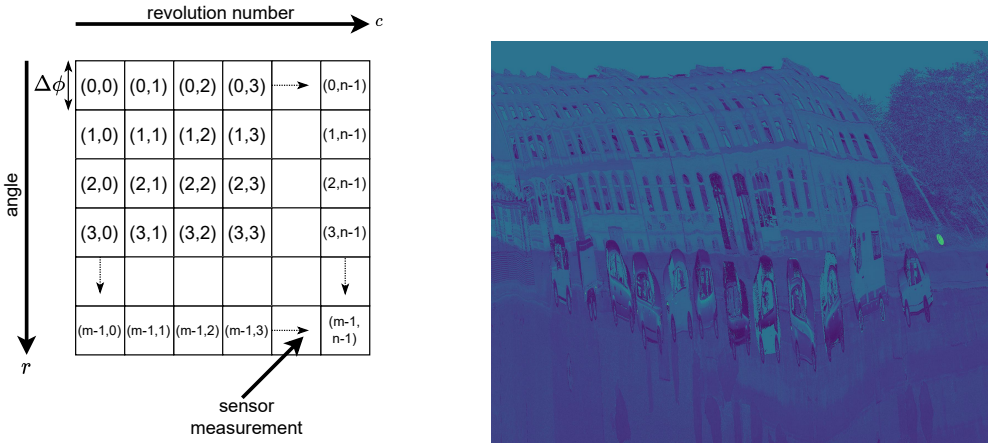
### Projections

By projecting a 3D point into a (virtual) camera using Equation 2.4 the point can be mapped into a 2D grid. Depending on the application the distance between the camera center and the 3D point or the distance between a plane and the 3D point is calculated and mapped to the pixel, resulting in a "2.5D" image or height-map. Other properties of the 3D point such as reflectance or color can be stored in another channel of the image. A typical application of these images are digital elevation models (DEM), which represent the height of a 3D terrain surface. This data is often collected by airborne laser scanners and projected into a grid from which the height of the surface can be derived. A very simple way of projecting a 3D surface can be done by **orthographic projection**, where a 3D point $v$ is mapped into a grid by $P_{ortho}$:

$$P_{ortho}v = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{2.11}$$

The value $v_z$ can be stored in the grid cell itself. If the 2.5D image has a sufficiently high grid resolution, it is less likely that more than one 3D point will be mapped in the same grid cell. This is usually not the case with other projections, such as perspective projection, where overlapping surfaces can be occluded.

Line scanners can also be projected without this type of occlusions by a **spherical projection**. In this case every revolution of the scanner head is appended along the x-axis of the image, Fig. 2.8a. Therefore, the y-axis is indexed by the angular increments of the scanner head $\Delta\phi$, each line being mapped to a specific azimuth angle. If the scanner is moved e.g. by a Mobile Mapping System, the scanstrip can produce a strongly morphed but still recognizable image, Fig. 2.8b. In this case, the objects are distorted by the ego motion of the car, which can result in elongated objects if the car is moving slowly, and shrunken objects if the car is moving fast. This type of data representation has the advantage over voxel grids that they are very dense and have almost no quantization errors. In contrast to a raw point cloud, the neighborhood information is roughly preserved in a scanstrip. However, it is not guaranteed that neighboring pixels are the closest points to each other.




*(a)* Scanstrip coordinate system

*(b)* Scanstrip colored by reflectance

*Figure 2.8: Example scanstrip captured with a line scanner mounted on a Mobile Mapping System (right) and the corresponding coordinate system (left)*

#### 2.1.2.2 Ray Tracing for Occlusion Determination

Rendering point clouds can be difficult because they are sparse and it is not clear how to fill empty areas and exclude occluded points, such as points behind walls and buildings. When individual 3D points on a surface are projected into an image, points behind the object may not be masked, resulting in a "transparent" looking object surface. It is often sufficient to enlarge the points to create the impression of a continuous surface. Splats can be rendered instead of points for this purpose. A splat is defined as an elliptical surface with a size determined by the local point density (Zwicker et al., 2001). Another way is to trace the ray between the camera center and the 3D point through space and estimate whether it hits a surface or not, which is therefore called ray tracing. Ray tracing is often used for very realistic image rendering, but in the case of point cloud visualization it can also be used to simulate occlusions. To estimate whether the ray will hit a surface, each point is magnified. Since ray tracing is computationally expensive because each ray must be traced through the space containing the point cloud, a voxel grid can be superimposed because finding a neighboring voxel only has a time complexity of $\mathcal{O}(1)$ and is therefore much faster than searching through the entire point cloud. The voxel also serves as a rough approximation to object surfaces by wrapping each point with a box volume. So the real test is not whether the ray intersects the point or surface, but only whether it hits the wrapping volume. To traverse the voxel grid, Bresenham's line algorithm can be used (Bresenham, 1965). This is a line drawing algorithm for determining grid cells that should be chosen to form a straight line between two points in the grid.

Occlusions can be detected as follows: All points of a point cloud are sorted into a voxel grid. The ray is then traced to each camera center in this grid, and if an occupied cell is found along the ray, the point is considered as occluded. In order to speed up the calculation, a voxel pyramid can be implemented that uses voxel grids with different resolutions to avoid excessive traversal of empty regions. The ray is first drawn through the voxel grid with the largest box sizes to check whether it might hit an occupied cell. As soon as it finds an occupied voxel, the voxel grid with the next higher resolution are added to the queue. This is repeated until the voxel grid with the highest resolution is reached or no more occupied voxels are found.

#### 2.1.2.3 Region Growing

Region growing is used to find segments in images. If neighboring points in an image meet certain homogeneity criteria, they belong together and form a segment. In contrast to e.g. thresholding, region growing uses the spatial information of an image. The seeded region growing by Adams and Bischof (1994) can in general be described by the following three steps:

1. Choose a seed pixel.
2. Check the neighboring pixels and add them to the region if they meet the homogeneity criterion.
3. Repeat the second step for each newly added pixels; stop if no more new pixels can be added.

The initial regions start at the positions of the seed pixels. Therefore, the choice of the seed pixels is very important, since they significantly influence the result. Depending on the problem, the seed regions can be selected manually, within a certain gray level range, e.g. if light or dark segments are to be found, they can also be selected randomly or placed uniformly on a grid. The choice of the homogeneity criterion is also crucial for moderate success. It can be based on any properties of the regions in the image, such as color, intensity, grayscale, or variance.

The algorithm can be easily extended to 3D point clouds. In this case adjacent points are found by a function, e.g. the Euclidean distance. The homogeneity criterion must also be adapted to the problem. The method developed by Rabbani et al. (2006) uses a smoothness constraint to find neighboring regions. The following pseudocode describes the algorithm in detail, it is presented as implemented in the Point Cloud Library[2] (PCL) of Rusu and Cousins (2011):

**Algorithm 1** 3D region growing segmentation

**Require:** Point cloud $P$, Point normals $N$, Point curvatures $cur$, neighbour finding function $\omega(.)$, curvature threshold $c_{th}$, angle threshold $\theta_{th}$

```
1:  R = [ ]: Empty region list
2:  A = [1, ··· ,|P|]: List of available points
3:  procedure RegionGrow3D
4:      while A is not empty do
5:          R_c = [ ] #current region
6:          S_c = [ ] #current seeds
7:          P_min = min(P[A].cur) # point with min curvature in A
8:          S_c.append(P_min)
9:          R_c.append(P_min)
10:         A.remove(P_min)
11:         for i=0 to size(S_c) do
12:             B_c = ω(S_c[i])# Find nearest neighbours of current seed point
13:             for j=0 to size(B_c) do
14:                 if A.contains(B_c[j]) and cos^-1(|S_c[i].N, S_c[j].N|) < θ_th then
15:                     R_c.append(B_c[j])
16:                     A.remove(B_c[j])
17:                     if B_c[j].cur < c_th then
18:                         S_c.append(B_c[j])
19:         R.append(R_c) # Add current region to global region list
20:     return R
```

[2] https://pointclouds.org/

The algorithm first sorts the points by their estimated curvature value to find the seed points which are the points with the smallest curvature (line 7). The curvature value is a local feature that can be calculated by computing the eigenvalues $\lambda$ of the covariance matrix built from a point and its k-nearest neighbors. According to Rusu and Cousins (2011) the curvature can be estimated by the relationship between the eigenvalues of the covariance matrix as follows $\frac{\lambda_1}{\lambda_1+\lambda_2+\lambda_3}$ with $\lambda_1 < \lambda_2 < \lambda_3$. Points with low curvature are located in flat regions. As stated by Rusu and Cousins (2011), it is preferable to grow regions from these points, as this reduces the total number of segments.

For every seed point the nearest neighbours $B_c$ are looked up (line 12). The algorithm then checks for each neighbor the angle between its normal and the normal of the current seed point (line 14). If the angle is less than the threshold $\theta_{th}$, it is added to the current region (line 15). If the curvature *cur* of the current neighboring point $B_c[j]$ is below the curvature threshold $c_{th}$ it is added to the current seeds and thus extends the for loop (line 17-18). After all current seeds have been consumed the current region is added to the global region list (line 19). Then the same process is repeated until $A$ contains no more points. Finally the algorithm returns a list of regions in line 20.

## 2.2 Machine Learning Fundamentals

In this section, some basics of machine learning are briefly introduced, i.e. some types of learning are covered, basics of gradient-based learning and in particular, gradient-boosting decision trees are covered, which are later also used in experiments. A very comprehensive overview of machine learning without a focus on deep learning is provided by Murphy (2012). Alternatively Goodfellow et al. (2016) provided an overview especially for deep learning.

### 2.2.1 Types of Learning

There are a variety of learning types, but overall machine learning is usually divided into supervised and unsupervised learning (Murphy, 2012, p. 2). In **supervised learning**, the goal is to learn a function $F$ that maps from an input $x$ to an output $y$ given a **training dataset** that contains pairs of input and output (label or target) data. The input data can be of any type, such as a row vector of numbers relating to specific events per column, or something more complex such as an audio signal, image or point cloud. The key is that for every input in the dataset there is an expected result, hence the name "supervised", which is often also called a label, annotation or ground truth. Labels can be created by humans (experts) or are collected automatically with human supervision. To "learn" such a mapping, the trainable (changeable) parameters $\theta$ of the function $F$ are tuned. The general training protocol is to find good values for these parameters so that a distance function $\text{Loss}(y_i,\hat{y_i})$ between each prediction $F(x_i) = \hat{y_i}$ and ground truth $y_i$ is minimal. Depending on whether the label data is numerical or categorical, one refers to a **regression** or a **classification** problem.

The second type of learning is **unsupervised learning**. It describes the task of learning from a set without labels to find patterns in the data (Murphy, 2012, p.2). According to Murphy (2012, p.2), it is a less well-defined problem. However, popular unsupervised learning problems include **clustering**, where the goal is to find subsets (clusters) in datasets where each data point is associated with a cluster, similar to classification where subsets can be grouped by a class label (Murphy, 2012, p.10). Another popular unsupervised learning task is learning how to compress the data, i.e. reduce the dimensionality, by discovering **latent factors** that describe the data (Murphy, 2012, p. 11). There exists also **semi-supervised learning** which is a task that lies between a supervised and unsupervised setting. The goal is almost always to reduce expensive

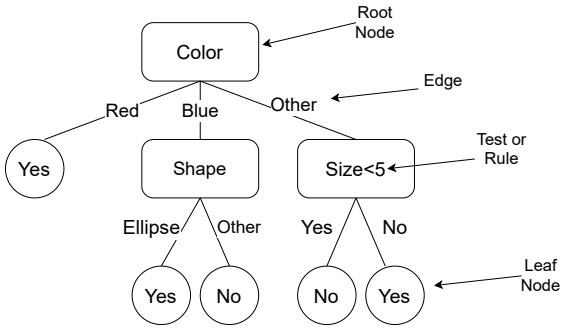



*Figure 2.9: Example of a very simple decision tree with three levels and three attributes (Color, Shape and Size). The tree is trained on a binary classification dataset and outputs either Yes (1) or No (0).*

annotation costs. In most cases a model is trained by using a large number of unlabelled instances to discover latent representations (unsupervised) in combination with a limited number of labelled instances (Van Engelen and Hoos, 2020). This topic will be discussed in more detail in Chapter 3.

### 2.2.2 Supervised Learning - Illustrated by Decision Trees

Since decision trees are trained in a supervised manner and are also used in this thesis, the following text does not only show briefly how to train a decision tree, but it also gives an introduction to the general procedure of training a supervised method. A decision tree is a method that makes a prediction by partitioning the input with simple decision rules based on thresholds or categories. In the following, it is assumed that the input data is categorical only, i.e. each input $x_i$ contains a list of attribute variables that can take one of a limited and fixed number of possible values. A decision tree has a hierarchical structure and consists of nodes and edges, s. Fig 2.9. Each node applies a test for the value of a certain attribute of the input. The edges of the node are connected to child nodes and correspond to each possible outcome of the test. A decision tree is traversed by starting at the root and comparing the attribute of an input sample with the attributes of the nodes. Depending on the result of the comparison, the tree is traversed to the corresponding sub-node, which tests another attribute, and so on. The final nodes of decision trees that have no child nodes are the leaf nodes. They store the outcome of the decision tree, which can be, for example, a histogram, a real value or a class number.

#### Kullback Leibler Divergence

As described in the introduction of this Section 2.2.1, in supervised learning a loss function is defined, and the goal is to optimize it. The **loss function** is sometimes referred to as a **cost** or **objective function** and measures the difference between predictions and labels. Similarly, for decision trees, the primary objective is to determine which attributes of the input should be considered in each node at each level. Depending on whether the decision tree is intended to solve a regression or a classification problem, the loss function may be, for example, the root mean square error (RMSE) or, in the latter case, the information gain (Murphy, 2012, p.547). The information gain is based on the entropy $H(X)$ in information theory, which measures the average level of "information" for a random variable $X$ with possible outcomes $\{x_1, \cdots, x_n\}$

$$H(X) = \sum_{i=1}^{n} -P(x_i) log\ P(x_i), \tag{2.12}$$

where $P(x_i)$ is the probability that $x_i$ occurs. This formula gives a measure of how uncertain the distribution in $X$ is. That is, $H(X)$ is zero if the set contains only one peak and zeros otherwise, and reaches the maximum value $log(n)$ if all probabilities are equal. In this way, entropy can be used to calculate how balanced the distribution of classes is in the dataset. An entropy of 0 indicates that the set contains only one class and an entropy of $log(n)$ suggest a balanced dataset.

Information gain $IG(X,A)$ measures the entropy of the dataset before and after a transformation, i.e., the reduction in entropy or surprise from splitting a dataset according to a particular value of a random variable. Here the entropy is used to calculate how the change to the dataset impacts the class distribution. A smaller entropy suggest less surprise and more purity. It is calculated by the entropy of the original set $H(X)$ minus the summation of the weighted entropy of each subset

$$IG(X,A) = H(X) - \sum_{v \in A} \frac{|X_v|}{|X|} H(X_v) = H(X) - H(X|A), \tag{2.13}$$

where $H(X|A)$ is the conditional entropy of $X$ given the value of the categorical attribute $A$. It is calculated by splitting the dataset into subsets for each possible value of $A$ and calculating the sum of the ratio multiplied by the the entropy of each subset. The ratio $\frac{|X_v|}{|X|}$ is calculated by dividing the number of the attributes $A$ that have the value $v$, by the number of all examples in the dataset. Moreover, $H(X_v)$ is the entropy of the subset of samples where $A$ has the value $v$.

The information gain is close to the Kullback-Leibler (KL) divergence which measures the "distance" between two probability distributions $P$ and $Q$ (Murphy, 2012, p.57f.). The KL-divergence for discrete probability distributions is defined as follows:

$$D_{KL}(P||Q) = \sum_{i=1}^{n} P(x_i) log \frac{P(x_i)}{Q(x_i)}, \tag{2.14}$$

where $P(x_i)$ and $Q(x_i)$ define the probability for a specific event $x_i$ to occur in both distributions and $n$ is the number of possible events of the random variable $X$. To express the information gain as KL-divergence one can use the joint distribution of both variables $P(X,A)$ and the probability distribution $P(X) \cdot P(A)$:

$$IG(X,A) = D_{KL}(P(X,A)||P(X) \cdot P(A)) \tag{2.15}$$

Here, the joint probability is the probability of occurrence of the intersection of $X$ and $A$. If both variables are independent, $P(X,A)$ is equal to $P(X) \cdot P(A)$ and the KL divergence between both distributions is zero. Therefore, maximizing the information gain is equal to minimizing the KL-divergence (Quinlan, 1986). It should be noted that the KL divergence is not a true distance measure because the function is asymmetric. For a symmetric version of the KL divergence, the Jensen-Shannon divergence can be used, which will not be discussed in this thesis (Lin, 1991).

#### Cross Entropy

The cross entropy $H(P,Q)$ has the property that its minima have the same location as the minima of the KL divergence $D_{KL}(P||Q)$.

$$H(P,Q) = \sum_{i=1}^{n} -P(x_i) log\ Q(x_i) = H(P) + D_{KL}(P||Q) \tag{2.16}$$

In supervised learning, $P$ refers to the (fixed) "true" distribution, which is given by the label $y$ and $Q$ is the estimated distribution, which is predicted by a classifier. Minimizing $H(P,Q)$ means

that the predicted probability distribution $Q$ has a peak at $max(P)$. Note that in a classification problem, where there is usually only one class per instance, the labels are "one-hot coded", which means that the distribution has only one peak of 1 at the label and is 0 otherwise.

### Building a Decision Tree

The following procedure shows how to build a tree using the ID3 algorithm presented by Quinlan (1986). Training a decision tree starts with the root node only. A tree is usually trained from top to bottom where the attributes that maximize the information gain are selected first.

**Algorithm 2** method to grow a decision tree

```
1:  procedure ID3(node, examples S)
2:      maxGain,bestA = 0,null
3:      for attribute a in S do # determine the attribute which leads to the largest IG
4:          gain = IG(S,a)
5:          if gain>maxGain then
6:              maxGain,bestA = gain,a
7:      for each value of bestA do
8:          Create a new child node
9:      subsets = Split(S,bestA) # split S into subsets for all possible values of bestA
10:     for each child node/subset do
11:         if not (subset is pure) then
12:             ID3(child node, subset)
```

The algorithm greedily searches for a test (bestA) that has the highest information gain (lines 3-6). To do this, $S$ is divided into subsets for the attribute bestA. Then, for each value that bestA can have, a child node is created so that the tree forms a new branch (lines 7-8). For each value, there is a subset containing the remaining attributes (line 9). The whole process is repeated (line 12) until the subset is pure (line 11), i.e. has no remaining attributes, or until the result is clearly defined.

Apart from that, the decision tree can have several **hyperparameters**, such as the maximum tree depth. These parameters can also be "tuned", meaning that there are many possible models that can be trained on the same data. The general problem this refers to is **model selection** (Murphy, 2012, p.22-24). To find the best possible model, the dataset is often split into training, validation, and test sets. Each model is fit to the training set and tested on the validation set. The test set, which was never part of the selection process, then serves as an unbiased estimate for the actual model performance. Depending on the algorithm, the validation set is sometimes included after the selection process and the best model is re-trained on both sets. In general, one wants a model that has a good performance and generalizes well to unseen data, i.e., that has similar performance on the training and test sets. This problem is known as the bias variance tradeoff.

### The Bias Variance Tradeoff

The bias variance tradeoff is a central problem in supervised learning defined by Geman et al. (1992). It is originally formulated for least squares regression, but similar decompositions have been found for classification problems, see (Domingos, 2000). The bias variance tradeoff is based on the fact that in supervised learning the mean square error (MSE) can be decomposed into variance and bias error:

$$\text{MSE} = \text{variance} + \text{bias}^2 \tag{2.17}$$

Ideally, a model is able to detect regularities in the training data while being able to generalise to unseen data. In reality, this is usually not the case, as a model that fits the training data too well or too poorly often does not generalise well to new data.

The bias error occurs when the model complexity is too low. It describes a wrong assumption in the trained model. The model is not able to capture the true meaning of a feature and cannot relate it to the result; the model is **underfitting**.

A large variance error, on the other hand, is related to a situation in which the model captures the training data too well because the model complexity is too high. It is able to capture even small fluctuations in the training set and fits the model to them almost perfectly. A high variance is related to the situation in which the model will **overfit** to the training data because it almost memorizes the training data and thus loses the possibility to generalize to new data.

Both errors are in conflict with each other, which means that a lower bias often increases the variance and vice versa. There are many approaches to address the problem. For example, the depth of a tree in decision trees or the number of neurons in an artificial neural network, which will be introduced in the next chapter, is directly related to the complexity of the model, which means that very deep trees or a high number of neurons leads to a high variance but low bias error. It is therefore necessary to find the optimal model complexity so that the model can generalise well.

### 2.2.3 Boosting

When many different decision trees are combined, they are called an ensemble or forest. Random forests, such as those described by Ho (1995), are trained using bagging, which means that each decision tree is trained in parallel on random subsamples or random features of the original training data, resulting in different decision trees. Together, they form an ensemble that can classify an input sample by summing or averaging the different predictions of multiple trees.

Boosting is another type of meta-learner that wraps a base learner algorithm to combine many (weak) models into one strong model. The original boosting algorithm (AdaBoost) was invented by Freund and Schapire (1996). Unlike, for example, random forests, the boosting algorithm works iteratively by adding a new trainable classifier $f_t(x)$ at each iteration step $t \in \{1,\ldots,T\}$. All classifiers are combined by summing their outputs to form the final prediction of the ensemble $F_T(x)$:

$$F_T(x) = \sum_{t=1}^{T} \alpha_t f_t(x) \tag{2.18}$$

At each boosting step $t$ a new weak classifier is added that outputs a weighted prediction $\alpha_t f_t(x)$. The new classifier is minimized together with the previously trained ensemble $F_{t-1}$

$$\mathcal{L}_t = \sum_i Loss(y_i, F_{t-1}(x_i) + \alpha_t f_t(x_i)), \tag{2.19}$$

where $Loss(y_i,\hat{y}_i))$ is a function that calculates the error between the label $y_i$ and the prediction $\hat{y}_i$ with $i \in \{1,\ldots,N\}$ training samples and the learning rate $\alpha_t$ is step $t$. An important part of the algorithm is that at each iteration $t$ a weight $w_{i,t}$ is assigned to each training sample $(x_i,y_i)$ which is calculated based on the current error $Loss(F_{t-1}(x_i))$ on that sample. The weights augment the training examples at each training step in such a way that more attention is put on incorrect predictions. This means that the new added weak learner attempts to correct the errors of its predecessors.

A more advanced version of Adaboost is gradient boosting invented by Breiman (1997) and Friedman (2002). This algorithm is later used in this thesis in the form of Gradient Boosted Decision Trees (GBDT). Gradient boosting differs from Adaboost in the way the weights are calculated at each training step. Instead of weighting the training samples, in gradient boosting the leaner $f_t$ is fit to the residuals $r_t$ of $F_{t-1}$:

$$r_t(x_i) = \frac{\partial Loss(y_i, F_{t-1}(x_i))}{\partial F_{t-1}(x_i)} \tag{2.20}$$

which means that instead of fitting each new weak learner $f_t$ to the weighted dataset like in Adaboost the learner is fitted to the gradient with respect to the current prediction $F_{t-1}$. The motivation is that this procedure follows the residuals, i.e. mistakes made by the ensemble by using steepest gradient descent.

## 2.3 Deep Learning

In this section the basics of artificial neural networks are explained in detail, followed by an introduction to deep learning including 2D and 3D data processing and generative adversarial networks.

### 2.3.1 Basics

The Artificial Neural Network (ANN) is inspired by structure of the human nervous system. The Perceptron developed by Frank Rosenblatt in 1958 can be seen as the first ANN and is represented by a simple mathematical function (Rosenblatt, 1958). It consists of an input vector $x$, a fixed nonlinear (activation) function $\psi$ and a (trainable) parameter matrix $W$ containing weights $w_{i,j}$ and biases $b_i$.

$$\hat{y} = \psi(Wx) \tag{2.21}$$

The parameter matrix $W$ and the input vector $x$ are defined as follows

$$W = \begin{pmatrix} b_1 & w_{1,1} & \cdots & w_{1,n} \\ b_2 & w_{2,1} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_m & w_{m,1} & \cdots & w_{m,n} \end{pmatrix}, x = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \tag{2.22}$$

Equation 2.21 is a compact notation for a function of type $\hat{y} = f(\sum w_i \cdot x_i + b_i)$.

Training, i.e., the determination of $w$ is based on minimizing the distance between the output $\hat{y}$ and an observation $y$. In order to measure the distance there exists a wide range of different functions which are chosen based on the problem at hand. An example for such a function is the Euclidean distance, also referred to as L2-Norm:

$$\mathcal{L} = ||y - \hat{y}||_2 = \sqrt{\sum (y - \hat{y})^2} \tag{2.23}$$

$\mathcal{L}$ can be minimized by calculating the partial derivative $\frac{\partial \mathcal{L}}{\partial w}$ and following the gradient to a (local) minimum. This method is called gradient descent. Gradient descent is an iterative procedure in order to update the (randomly initialized) weights.

$$w_{i,j}(t+1) = w_{i,j}(t) - \lambda \frac{\partial \mathcal{L}}{\partial w_{i,j}(t)}, \tag{2.24}$$

where $w_{i,j}(t)$ denotes the weight at iteration step $t$ and $\lambda$ is the "learning rate" which defines the distance the vector moves in each step. The number of steps depends on the selected criterion and can be a fixed number, or until the updates become very small, or a more complex criterion.

#### Multilayer Perceptron

As early as 1969, Marvin Minski and Seymour Papert criticised the perceptron not to have the ability to represent a logical XOR and that it was very complicated for the computers of the time to train these networks (Papert and Minsky, 1969). The XOR problem can be solved by adding further layers which is therefore called multilayer perceptron (MLP), but these layers could not be trained before the development of the backpropagation algorithm.

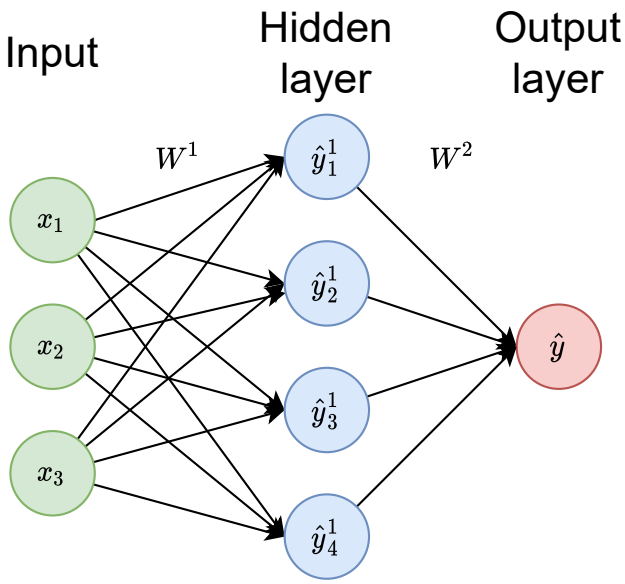



*Figure 2.10: Example of a simple multilayer perceptron (MLP) with two fully connected layers and two weight matrices $W^1$ and $W^2$.*

A multilayer perceptron as visualized in Figure 2.10 can consist of several consecutive layers $p \in \{0,\dots,P\}$. Each circle refers to a node or "neuron" and each line between the nodes corresponds to a trainable parameters referred to as weight. As every node of one layer is connected to the nodes of the neighboring layers, they are referred to as "dense" or "fully connected" layers. The output $\hat{y}^{(p)}$ for one layer is calculated in the exactly same way as in Equation 2.21:

$$\hat{y}^{(p)} = \psi^{(p)}(W^{(p)}\hat{y}^{(p-1)}), \tag{2.25}$$

where $\hat{y}^{(p-1)}$ denotes the output of the previous layer and $\hat{y}^{(0)} = x$. By propagating the input vector $x$ through the consecutive layers, the final output $\hat{y}^{(P)}$ can be obtained:

$$\hat{y}^{(P)} = \psi^{(P)}(W^{(P)}\psi^{(P-1)}(\dots W^{(2)}\psi^{(1)}(W^{(1)}x)\dots)) \tag{2.26}$$

As soon as a large number of hidden layers are used, one can generally speak of a deep neural networks and the learning procedure can be called deep learning.

### Backpropagation

The backpropagation algorithm is used to calculate the gradient for every weight in every layer (Rumelhart et al., 1986). Backpropagation starts with a loss function *Loss* calculating the error between the label data $y$ and the output of the last layer $\hat{y}^{(P)}$:

$$\mathcal{L} = \text{Loss}(y,\hat{y}^{(P)}) \tag{2.27}$$

The goal is to calculate an update for every weight $w_{i,j}^{(p)}$ so that the error is minimized by using an optimizer such as stochastic gradient descent. To do this, Equation 2.26 is rewritten as follows:

$$\sigma_i^{(p)} = \sum_{j=1}^{n^{(p)}} w_{ij}^{(p)} \hat{y}_j^{(p-1)} = \sum_{j=1}^{n^{(p)}} w_{ij}^{(p)} x_j^{(p)} \tag{2.28a}$$

$$\hat{y}_i^{(p)} = \psi(\sigma_i^{(p)}) \tag{2.28b}$$

It can be seen from the equations that the input of layer $p$ is the output of layer $p-1$, respectively $x^{(p)} = \hat{y}^{(p-1)}$. Furthermore, $i \in \{1,\ldots,m^{(p)}\}$ and $j \in \{1,\ldots,n^{(p)}\}$ are the row and column index respectively, of the weight matrix $W^{(p)}$, with size $m^{(p)} \times n^{(p)}$ of layer $p$. This means that the matrix size can vary between the layers. However, for successive layers to be compatible, $n^{(p)} = m^{(p-1)}$ must hold.

For gradient descent the partial derivative of the error function with respect to the weight $\frac{\partial \mathcal{L}}{\partial w_{ij}^{(p)}}$ is calculated. By first applying the chain rule and then substituting Equation 2.28a, the partial derivative is rewritten as follows:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(p)}} = \frac{\partial \mathcal{L}}{\partial \sigma_i^{(p)}} \frac{\partial \sigma_i^{(p)}}{\partial w_{ij}^{(p)}} = \delta_i^{(p)} \frac{\partial}{\partial w_{ij}^{(p)}} \sum_{j=1}^{n^{(p)}} w_{ij}^{(p)} y_j^{(p-1)}, \tag{2.29}$$

where $\delta^{(p)}$ is shorthand for the error at layer $p$. For the calculation of the $\delta^{(p)}$ it must be distinguished whether $\delta^{(p)}$ must be determined at the output layer of the network using $\hat{y}^{(P)}$ (case 1.) or in a hidden layer with $p < P$ (case 2.): After the forward pass, the backpropagation starts by calculating the gradient in the output layer, which is the **first case** where $p = P$ (Goodfellow et al., 2016, p.206). The update is here calculated as follows:

$$\delta^{(P)} = \frac{\partial \hat{y}_i^{(P)}}{\partial \sigma_i^{(P)}} \cdot \frac{\text{Loss}(y,\hat{y}^{(P)})}{\partial \hat{y}^{(P)}} \tag{2.30}$$

In the **second case** where $p < P$ the gradient is calculated as follows:

$$\delta_i^{(p)} = \psi'(\sigma^{(p)}) \sum_{j=1}^{n^{(p+1)}} \delta_j^{(p+1)} w_{ji}^{(p+1)} = \frac{\partial \hat{y}_i^{(p)}}{\partial \sigma_i^{(p)}} \sum_{j=1}^{n^{(p+1)}} \delta_j^{(p+1)} w_{ji}^{(p+1)} \tag{2.31}$$

#### 2.3.1.1 Training an MLP

To complete the basics of artificial neural networks, this section deals with the basic principle of training an ANN. To train an ANN in a supervised way, the following requirements must be met (Goodfellow et al., 2016, Chapter 5.10):

- The **architecture** of the network $f(x;\theta)$ must be defined, whereby $\theta$ is the set of trainable variables and $f(x)$ is a function that represents the full forward pass of the ANN.
- A **dataset** $D$ must be available containing tuples of input data $x$ and the corresponding labels $y$.
- A **loss function** $\mathcal{L}$ must be defined that measures the distance between the prediction $\hat{y}$ and the label $y$.
- A **optimizer** must be chosen that calculates the updates for the trainable variables.

These requirements are the same for most ANNs, also in deep learning. The general training procedure for an ANN is as follows:

**Algorithm 3** Training an MLP with gradient descent

**Require:** network $f(x;\theta)$ and dataset $D$
  Initialise all weights $\theta$ with random values
  **procedure** TRAINING
    **while** not StopCriterion **do**
      **for** every sample $x_i$,$y_i$ in D **do**
        $\hat{y}_i = f(x_i)$ # forward pass
        $\delta^{(P)} = \psi'(\sigma^{(P)}) \cdot \nabla_{\hat{y}} \text{Loss}(y,\hat{y}^{(P)})$ #backward pass first case
        **for** $p = P-1,\ldots,1$ **do**
          **for** every Neuron $i$ in layer $p$ **do**
            $\delta_i^{(p)} = \psi'(\sigma_i^{(p)}) \sum_{j=1}^{n^{(p+1)}} \delta_j^{(p+1)} w_{ji}^{(p+1)}$ # backward pass second case
        **for** every weight $w_{ji}^{(p)}$ **do**
          $w_{ji}^{(p)} = w_{ji}^{(p)} - \eta \delta_j^{(p)} y_i^{(p-1)}$ # optimizer

Possible termination criteria for this algorithm are, for example, to stop after a pre-defined number of iterations, or when the loss is below a certain threshold. In reality, it is very common to measure not only the training loss but also the validation loss. Often the weights are stored at each validation step and the weights that reach the lowest validation loss are used for testing later.

In Algorithm 3 the model is trained with **gradient descent**. In this case, the training process is deterministic, as the gradient step is always calculated on the entire dataset. Otherwise, when the model is trained only on a smaller, randomly selected subset, it is called a **stochastic gradient descent** (SGD). In real situations, there may be billions of trainable model parameters and training examples. For additive cost functions, gradient descent requires computing the sum over all distances, which has a computational cost of $\mathcal{O}(m)$, where $m$ is the size of the training set. As the training set grows, the computation of a single gradient step may become too long for the model to converge in a reasonable time. Stochastic minibatch gradient descent computes an approximation of the expected gradient step using a **minibatch** of uniformly sampled training samples. The minibatch size $k$ is often set to a small number between one and a few hundred samples. This results in a slightly different equation than Equation 2.24 using the minibatch size $k$:

$$w(t+1) = w(t) - \frac{\lambda}{k} \sum_{i=1}^{k} \nabla_w \mathcal{L}_i \tag{2.32}$$

In **classification problems**, the network output and label vector and the loss function are usually adapted (Goodfellow et al., 2016, p. 173-180). In this case, the output vector $\hat{y}_i$ and the label vector $y_i$ have the length $C$ with is the size of all possible classes. Typically, $y_i$ is one-hot coded. To represent the output as valid probability distribution it is usually normalised so that all values in $\hat{y}_i$ are between 0 and 1 and $\sum_{j=1}^{C} \hat{y}_{i,j} = 1$. For this purpose the softmax function can be used:

$$\text{softmax}(z)_{i,j} = \frac{exp(z_{i,j})}{\sum_{k=1}^{C} exp(z_{i,k})} = \hat{y}_i \tag{2.33}$$

The vector $z_i$ is the output $\sigma^{(P)}$ of the last layer $P$. It represents the unnormalized log probabilities and is therefore often referred to as “logit”.

### Convolutional Layers

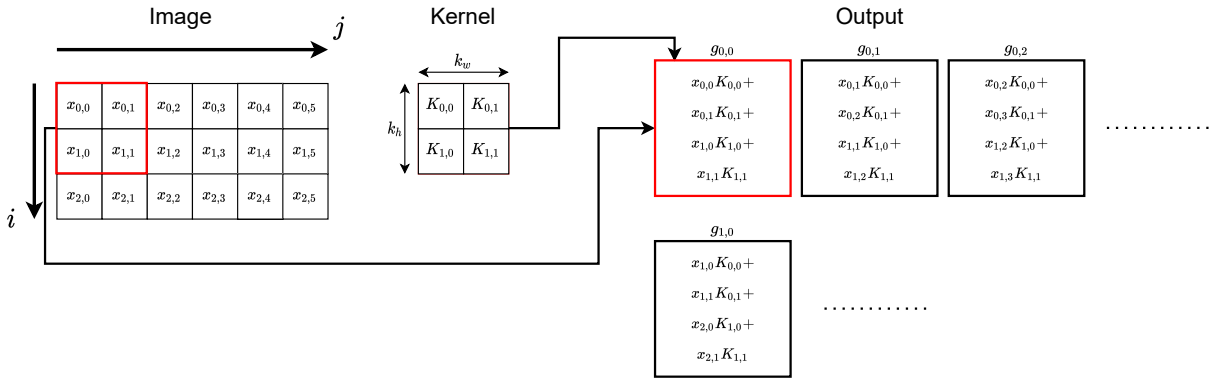



*Figure 2.11: An example for a 2D convolution without padding. The input is* $3 \times 6 \times 1$*, the kernel size* $2 \times 2 \times 1 \times 1$ *and the stride is* $s_1 = 1, s_2 = 1$*. The output* $g$ *has the size* $2 \times 3 \times 1$

According to the universal approximation theorem a network with one hidden "dense" layer is sufficient to approximate arbitrary real-valued continuous functions (Hornik, 1991). The problem is that these layers have high computational complexity, are fixed in size and are more prone to overfit to the data.

A convolution layer (LeCun, 1989) drastically reduces the number of weights by sharing them over the input (weight sharing) (Goodfellow et al., 2016, p.247, p.324f.). This is done by sliding a filter matrix (kernel) over the input and convolving it with the data, as in Fig. 2.11. The kernel contains the trainable weights of the neural network. Typical hyperparameters for a convolutional layer are the number and size of the kernels per layer, whether zeros are padded at the border of the input, the **dilation** that defines the distance between each kernel element, and the **stride** that determines how many values are skipped at each step. The output of each layer is called the **feature map** because it represents the detected features like a map at each specific input location for each kernel. Increased striding has the advantage of reducing the size of the feature map, which reduces computational complexity, but it also reduces the resolution of the location of the detected features. Finally, **zero padding** is used to increase the input size by adding zeros at the border, to control the output size of the convolution. Usually this is done so that the resulting feature map has the same size as the input.

For a 2D image with one channel, a kernel is defined by a 2-dimensional matrix. Since a layer often has more than one channel as input and uses many kernels, the filters $K$ are defined by a 4-dimensional matrix. The dimensions are given by $k_w \times k_h \times c_{in} \times c_{out}$, where $k_w$ and $k_h$ define the width and height of the kernels, $c_{in}$ is the channel dimension of the previous layer and $c_{out}$ the number of output features and kernels, respectively. The feature map $g$ is then calculated as follows:

$$g_{i,j,k} = \sum_{di,dj,q} x_{s_1 \cdot i + di, s_2 \cdot j + dj, q} \cdot K_{di,dj,q,k}, \tag{2.34}$$

where $x$ is the input matrix and $s_1$ and $s_2$ are the striding in the first and second dimensions, respectively. Obviously the convolution of higher dimensional data follows the same procedure but increases the number of trainable parameters significantly. As modern Graphics Processing Units (GPUs) are optimized for matrix algebra, convolutions are implemented in cuDNN[3] using matrix multiplication (Chetlur et al., 2014). This can be done by reshaping the kernel into a matrix $K_m$ with dimensions $c_{out} \times c_{in} k_h k_w$ and generating an input matrix by duplicating the data into a

[3] cuDNN is a common cuda library used in most deep learning frameworks

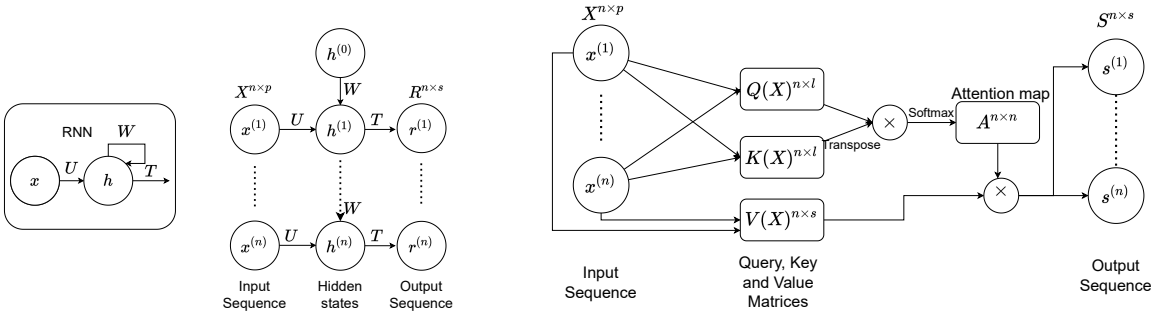
matrix $D$ with dimension $c_{in}k_hk_w \times NPQ$, where $N$ defines the minibatch size and $P$ and $Q$ the output height and width which depend on the chosen hyperparameters such as striding or padding for the convolution. The convolution can then be performed with a single matrix multiplication to form an output matrix $G_m$ with dimension $c_{out} \times NPQ$

$$G_m = K_m D \tag{2.35}$$

The output of this multiplication has the size of the number of features $c_{out}$ times the output size, which is equivalent to Equation 2.34. For further processing $G_m$ is reshaped to the actual size of the feature map $N \times P \times Q \times c_{out}$. The actual training of a network using convolutions is equivalent to an MLP that uses fully-connected layers (Goodfellow et al., 2016, p.345).

Convolutional layers are only capable of maintaining or reducing the spatial dimension of the input, but in some cases it is desirable to map from a lower resolution to a higher one, for example to reverse the effect of a strided convolution. For this purpose, **transposed convolutions** are used, also called "fractional strided convolutions" or "deconvolutions". Transposed convolutions follow a principle similar to that expressed in Equation 2.35. However, unlike regular convolution, transposed convolution broadcasts the input elements through the kernel by exchanging the forward propagation function and the backward propagation function of the convolution layer.

### 2.3.2 Self-Attention




*(a)* Left: simple RNN (recurrent neural network) corresponding to Equation 2.37. Right: Mapping a sequence $X$ to output $R$ with RNN

*(b)* Mapping a sequence $X$ to output $S$ with Self-Attention.

*Figure 2.12: Mapping an input sequence $X$ of length $n$ to the output sequences $R$ and $S$ of size $n \times s$. On the left an RNN is shown that maps from an input to a hidden state with matrix $U$, from hidden to hidden states with $W$, and from the hidden states to the output with matrix $T$. On the right, each element of the sequence $X$ is mapped independently to the matrices $Q$, $K$, and $V$ using three MLPs. $Q$ and $K$ are used to calculate an attention map $A$ that relates all entries in $V$, resulting in the output $S$. Both methods are similar in the way that they relate input sequences to each other. However, they do not produce the same results and are not equivalent to each other*

In the context of deep learning, attention refers to highlighting elements such as embeddings or words (Sutskever et al., 2014) or image regions (Kosiorek et al., 2017). In a mathematical sense, attention refers to the weighting of entities, e.g. calculating the weighted sum of a list of elements. Here, high weights correspond to something more important, i.e. an entity receiving more attention. There are many applications for attention mechanism. They were introduced in natural language processing (NLP) to highlight words that are useful for translating a sentence, while other words such as filler words or articles should receive less attention in this context (Sutskever et al., 2014).

These models used recurrent layers which map an input sequence to an output sequence, which can be expressed as follows:

$$h^{(t)} = \phi(b + Wh^{(t-1)} + Ux^{(t)}) \tag{2.36}$$

$$r^{(t)} = c + Th^{(t)} \tag{2.37}$$

Here $W$,$U$ and $T$ are the trainable parameter matrices and $x^{(t)}$ is the current input from a sequence $t \in \{1,\ldots,n\}$ and $h^{(t-1)}$ is the hidden state from a previous sequence element $t-1$. The output $r^{(t)}$ is formed from the hidden states for each input element $r$, see Fig. 2.12a. The advantage of using a recurrent layer is that it can relate objects within a sequence by providing information from previously seen elements. This property comes with many disadvantages, such as low training stability due to vanishing or exploding gradients in training. The problem is that the hidden state is a fixed-size vector that has a limited capacity. Sutskever et al. (2014); Bahdanau et al. (2015); Luong et al. (2015) tried to address this problem by weighting the hidden states with attention to remove unnecessary information. However, by design, Recurrent Neural Networks (RNNs) are slower than, for example, CNNs because they must be processed sequentially, since subsequent steps depend on previous steps. In comparison, in a convolutional layer, each convolution can be processed completely independently. Vaswani et al. (2017) proposed a so-called Transformer Network using self-attention to solve these problems. Self-attention was used to completely replace RNNs with an element-by-element operation, similar to convolutions or MLPs.

**Self-attention** relates each entity to every other one in parallel by creating a global attention map, see Fig. 2.12b. Given a collection of elements $x^{(t)}$ with $t \in \{1,\ldots,n\}$ to relate all entities, they are projected (linearly) to a $l-$dimensional query matrix $Q(x)$ and a key matrix $K(x)$. Thus, the channel dimension (column) of $Q(x)$ and $K(x)$ is and the number of rows correspond to the number of elements $n$ of the sequence. The relevance of the individual elements with respect to each other is measured by a matrix $A$, that can be obtained by applying the softmax function to the prediction of $Q(x)$ and $K(x)^T$.

$$A = softmax(Q(x)K(x)^T) \tag{2.38}$$

The matrix $A$ has size $(n \times n)$ and is normalized row-wise by the softmax function, i.e. so that each row sums up to 1. In most applications the attention scores are scaled by the square root of the query and key dimension $\sqrt{l}$. The output of the self-attention layer $S(x)$ is computed by the dot product between $A(x)$ and $V(x)$. Here, $V(x)$ is a function that controls the feature size of the output by projecting each element in the sequence of $x$ into an $s$-dimensional feature space, yielding the matrix $S$ of size $(n \times s)$.

$$S = softmax(\frac{Q(x)K(x)^T}{\sqrt{l}})V(x) = AV(x) \tag{2.39}$$

In $S$, each row contains the weighted sum over all elements of $V(x)$. By using the softmax function, each row of $A$ contains a distribution of weights, indicating how much each row of $V(x)$ should be considered to produce an element of the output sequence. During training, the weights in the functions $Q(x)$, $K(x)$ and $V(x)$ are adjusted. A network using such a layer is able to relate the elements of the input sequence.

In summary, the self-attention layer obtains an uncorrelated list of feature vectors and generates a correlated list of elements of the same length by calculating, for each feature vector, how strongly to consider all other elements in order to aggregate them by using the weighted sum. The benefit of doing this an improvement of the computational speed over RNNs and that no hidden-states are needed.

### 2.3.3 Generative Adversarial Networks

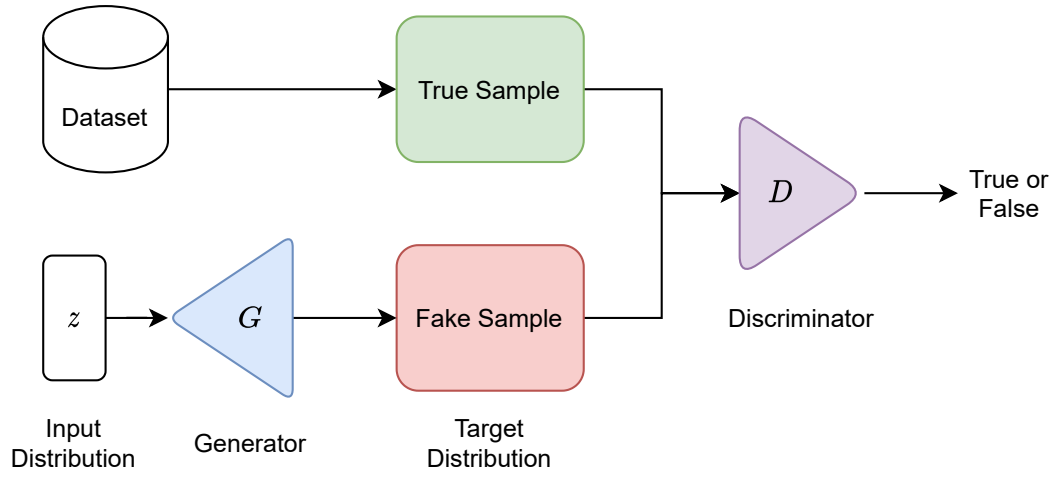



*Figure 2.13: The general structure of generative adversarial networks. The generator learns the mapping $x = G(z)$ and the discriminator learns to classify whether $G(z)$ and $x$ are real or generated.*

Generative Adversarial Network (GAN) were invented by Goodfellow et al. (2014). In its original form, a GAN is trained in an unsupervised way and learns to estimate a function that maps from one distribution to another. A GAN consists of a **generator network** that produce samples $x = G(z)$ and a **discriminator network** $y = D(x)$ that classifies the samples if they were drawn from the real distribution (class real) or if they were produced by the generator (class fake), see Fig. 2.13. The two networks are trained together so that the discriminator acts as a loss function for the generator, forcing it to produce samples that are indistinguishable from real data. Therefore, the generated instances also become negative training examples for the discriminator and the discriminator penalizes the generator for producing implausible data. Assuming that the discriminator produces a probability between zero (fake) and one (real), the following function describes the optimization problem:

$$\max_D \min_G V(D,G) = \mathbb{E}_{x\sim p_{data(x)}}[log(D(x)] + \mathbb{E}_{z\sim p_z(x)}[log(1 - D(G(z))] \tag{2.40}$$

Both networks are trained in a mini-max game in which the generator tries to minimize Equation 2.40 and the discriminator tries to maximize it. As the generator has no direct influence on the term $log(D(x))$, Equation 2.40 can be expressed using the following two loss functions:

$$\begin{aligned} \min_G \mathcal{L}(G) &= \frac{1}{m}\sum_{i=1}^{m} -log(D(G(z_i)) \\ \min_D \mathcal{L}(D) &= \frac{1}{m}\sum_{i=1}^{m} -log(D(x_i)) - log(1 - D(G(z_i)) \end{aligned} \tag{2.41}$$

The discriminator should learn to classify whether the input is real or synthetically generated by $G(z)$. The function $\mathcal{L}(G)$ shows that the generator $G(z)$ is optimised by generating samples such that $D(x)$ predicts values close to one. This means that the generator should produce data that is considered real by the discriminator. The discriminator, on the other hand, is optimised by generating values close to zero for $D(G(z_i))$ while predicting values close to one for $D(x_i)$. The picture in 2.13 shows the structure of the whole system. A typical GAN alternately trains the discriminator and the generator. During generator training which is based on $\min_G \mathcal{L}(G)$, the gradients propagate through the discriminator to the generator (although the discriminator does not update its weights during generator training). Therefore the weights in the discriminator

network influence the updates in the generator network, which also means that the discriminator must be differentiable. The algorithm of training a GAN is shown in Algorithm 4.

**Algorithm 4** General procedure to train a GAN

**Require:** generator $G(z;\theta^{(G)})$, discriminator $D(x;\theta^{(D)})$ and dataset $X$
  Initialise all weights $\theta^{(G)}$ and $\theta^{(D)}$ with random values
  **procedure** TRAINING
    **while** not StopCriterion **do**
      Sample minibatch of m noise samples $\{z_1,\ldots,z_m\}$ from prior distribution
      Sample minibatch of m samples $\{x_1,\ldots,x_m\}$ from dataset
      Generate minibatch of m fake samples from generator
      Update the discriminator by minimizing $\mathcal{L}(\mathcal{D})$ using gradient descent
      Update the generator by minimizing $\mathcal{L}(\mathcal{G})$ using gradient descent

In Algorithm 4 the GAN is trained by alternating between discriminator and generator update. However, there are different versions, For instance, were one network can be trained more frequently than the other Goodfellow et al. (2014). Apart from that the update of both networks is done using Algorithm 3, the choice of architecture and optimizer are hyperparameters that affect the training stability. As both networks compete with each other, training stability means that one network does not tend to outperform the other, generating either examples that cannot be detected as fakes or a discriminator that is always 100% accurate. A GAN converges when the discriminator and the generator reach a Nash equilibrium. This means that neither the discriminator nor the generator can locally improve their objectives. Since the invention of GANs, many different variants have been proposed in an attempt to increase the training stability and the quality of the prediction (Mao et al., 2017). Many different variants have also been introduced, such as conditional adversarial generative networks (Isola et al., 2017), which are discussed in the next chapter.

# 3 Related Work

In this chapter, all work related to this thesis is presented. Since the thesis is mainly concerned with semantic segmentation, the state-of-the-art models, especially those that will be used and compared to later, are presented in more detail. Other related topics covered and referenced are 3D semantic segmentation networks, semi-supervised learning, transfer learning, multi-view data, conditional adversarial networks and shape completion.

## 3.1 Classification and Semantic Segmentation (2D)

In 1989 Yann LeCun et al. trained a small convolutional neural network (CNN) with three hidden layers using backpropagation to recognize handwritten postal codes (LeCun et al., 1989). The input of the network is a normalized image with the size of $16 \times 16$ pixels and the output vector consists of 10 units (one per class). In the first and second hidden layers, convolutions were used instead of fully connected layers to reduce the number of trainable parameters, of which there are 1068 in the first layer and 2592 in the second layer. The third layer is fully connected to the second convolution and has 5790 parameters. The last layer, which outputs the class weights, is also fully connected and contains 310 trainable parameters, which results in a total of 9760 parameters of the CNN. The non linear function in each layer is a hyperbolic tangent, which limits the output range between -1 and 1. The network was trained using SGD, which took 3 days to converge on a Sun 4/260 workstation.

This example was not only the first solution for using a CNN with SGD to classify images, it also shows the general structure and procedure of how images are classified today in deep learning. Popular CNNs like AlexNet by Krizhevsky et al. (2012) or VGG16 by Simonyan and Zisserman (2015) are all similar in their structure. They first learn to encode image features using several stacked convolutions in order to reduce the resolution of the featuremaps and therefore learn a latent representation of the input. At the end, these features are fed to several dense layers which finally output the class distribution.

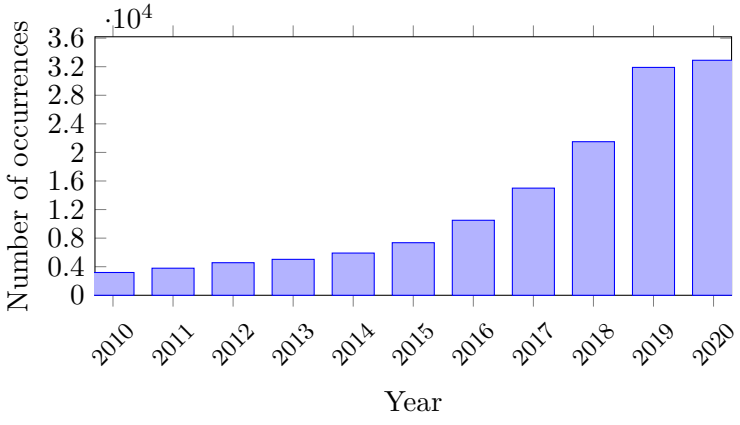
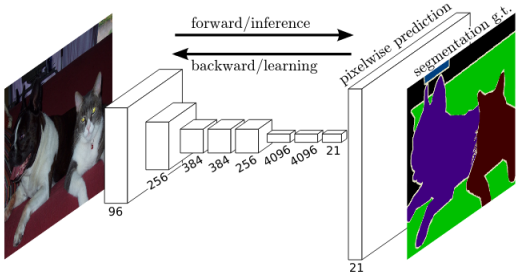
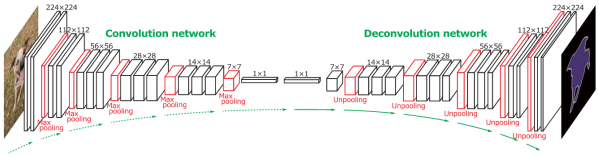



*Figure 3.1: Number of occurrences for the search term "semantic segmentation deep learning" returned by* `scholar.google.com` *for each year. The search excluded patents and citations.*




*(a)* FCN by Long et al. (2015). (Image source: Long et al. (2015))




*(b)* A network proposed by Noh et al. (2015). (Image source: Noh et al. (2015))

*Figure 3.2: Schematic overview of two semantic segmentation networks*

Nowadays in computer vision, **semantic segmentation** means the pixel-wise classification of an image. Its origin can be seen in classification (assigning only one label to an image). The terms origins from image segmentation, which is the general procedure of finding regions in images that belong together (Haralick and Shapiro, 1985). Typical methods are thresholding, Watershed-algorithm, Region Growing (Shih and Cheng, 2005), Graph-Cuts (Boykov and Funka-Lea, 2006) or even spatial clustering like K-means or DBSCAN (Shen et al., 2016). Later approaches used graphical models such as Markov- or Conditional-Random Fields (MRFs or CRFs). These methods segment an image into regions and create hand-crafted features from the individual and neighbouring image segments to predict a class for a given pixel or region (Xiao and Quan, 2009; Ladicky et al., 2010; Farabet et al., 2012; Kumar and Singh, 2012; Lerma and Kosecka, 2014).

A milestone in the field of semantic segmentation was reached with the introduction of the first fully convolutional neural network by Long et al. (2015) (Fig. 3.2a). It was introduced in 2015 after which deep learning together with semantic segmentation became increasingly popular, see Fig. 3.1. The basic idea of the Fully Convolutional Network (FCN) is that all its layers are convolutional. Traditionally, in classification, at least the last layer that outputs a single class distribution vector is fully connected, while FCN uses only convolutional layers to classify each pixel in the image. In FCN, the fully connected layer was instead replaced with transposed convolutions to produce a prediction with the same width and height of the input image. The authors had success in converting networks such as AlexNet by Krizhevsky et al. (2012), VGG by Simonyan and Zisserman (2015) or GoogLeNet by Szegedy et al. (2015) into FCNs by replacing their last layers by transposed convolutions. However, they all had the problem that the upsampling of the last convolutional layers seemed to be inaccurate. This happened because the spatial information was lost through multiple downsampling layers. The authors solved this problem by adding links between the intermediate and final layers to increase spatial accuracy, in other words, by transferring knowledge of “where” a particular object is located in the image from an earlier stage of the network. After that, the development of semantic neural segmentation networks became extremely popular, the field was driven by many applications in a short period of time which are going to be explained in the following.

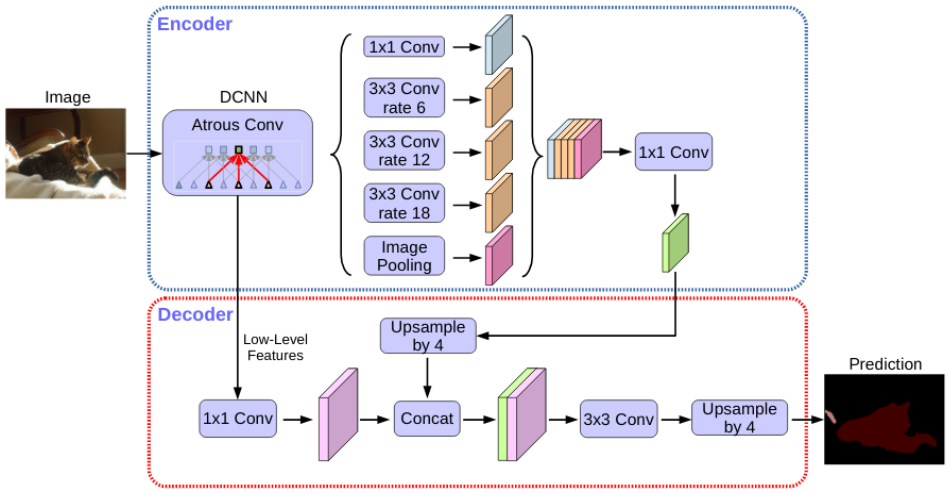



*Figure 3.3: Schematic overview of the Deeplabv3+ architecture. Image source: Chen et al. (2018)*

Often, the networks have common architectural structures, in particular each network can usually be divided into an encoder part and a decoder part:

- The **encoder**, sometimes called the "backbone", is the part that embeds the input into a high-dimensional and low-resolution feature map. Like in FCN, these are often classification networks such as AlexNet, VGG, GoogLeNet (Long et al., 2015; Noh et al., 2015).
- The task of the **decoder** is to map the high-level feature map from a lower resolution into the higher resolution pixel space in such a way that it returns a pixel-wise class distribution.

The encoder-decoder structure is often abstracted by an hourglass shape as already seen in Figure 3.2b. The reasons for such a design are manifold. First, it is computationally more efficient to progressively reduce the size of the feature map with each layer because it shortens the number of operations to be performed. Second, by applying pooling layers or striding, the network gains a higher receptive field and takes more context into account. Finally, the network tends to produce higher-order abstractions of objects because it is forced to describe them in a high-dimensional feature space but with lower resolution (Zeiler and Fergus, 2014). A rough analogy might be that the layer at the end of the encoder describes the scene, rather than exact assignments of each object to the pixels.

Recent semantic segmentation networks were often driven by newly discovered backends like ResNet by He et al. (2016), Xception by Chollet (2017) and Mobilenet by Sandler et al. (2018). A popular and widely adopted network is Deeplab in all its variants (Chen et al., 2018). Historically, the development of FCN and Deeplab can give an insight into the general direction other researchers were heading, therefore, these networks are explained in more detail. A main component of Deeplab is the dilated convolution, also referred to as atrous convolution. It was introduced in the first version of Deeplab by Chen et al. (2017b). The atrous convolution can be thought of as increasing the space between each convolutional kernel element. Mathematically it was already defined in Equation 2.34 by the value $d_i$.

The **first version of Deeplab** contained several advancements over earlier FCN models. Like FCN, the encoder is based on VGG and uses several max-pooling layers and striding to scale down the resulting feature map. To improve performance, they removed the last two max-pooling layers of the network and added dilated convolutions instead. The dilated convolutions increase the receptive field of the network without increasing the computational cost. Bilinear interpolation

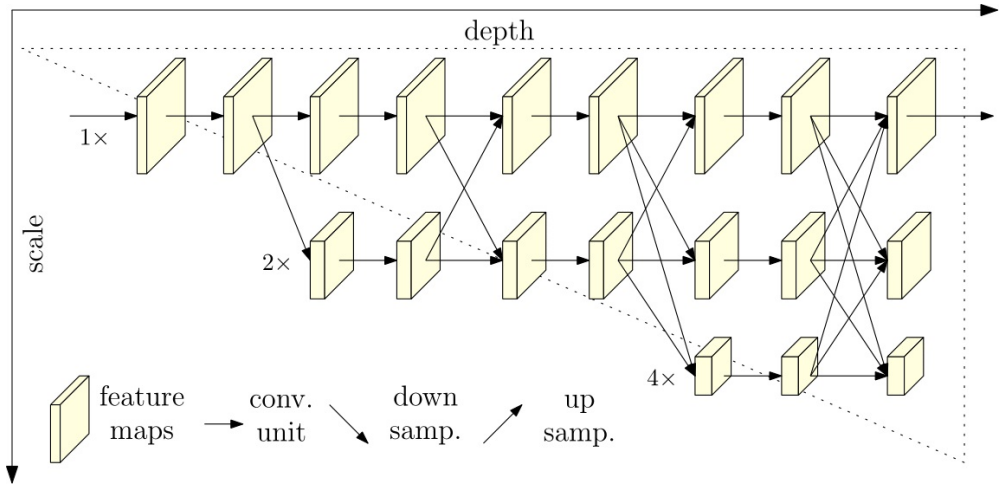



*Figure 3.4: Schematic overview of the connections in HRNet. Image source Sun et al. (2019).*

obviously requires much less computational effort and does not need to be trained. Finally, to improve the segmentation result, a fully connected Conditional Random Fields (CRF) layer was used. This is a layer that contains a CRF to incorporate smoothing terms that removes outlier predictions. This network later obtained an mIoU of 63.1 on Cityscapes dataset (Chen et al., 2017b)

The **second version of Deeplab** introduced **Atrous Spatial Pyramid Pooling** (ASPP) (Chen et al., 2017b). Since similar objects can appear in different sizes, the network should be invariant to scaling. Instead of rescaling the image and feeding it into the network to implicitly train object invariance, the authors created a network that has multiple parallel sub-branches that use atrous convolutions, each with a different dilation rate to account for different object sizes. These branches were later merged into a single feature map. This network achieved an mIoU of 70.4 on Cityscapes, which corresponds to an improvement of $\approx 7\%$ compared to Deeplabv1 (Chen et al., 2017b).

In the **third version of Deeplab** the authors mainly experimented with different ASPP modules (Chen et al., 2017a). They took into account that with higher dilation rates the valid image region becomes smaller (i.e. weights that are applied to non padded image regions). So instead of simply applying the ASPP module at the end of the encoder, they added skip connections from earlier layers to each output of the ASPP module branch. After that, the pyramid was fused in a way similar to Deeplabv2 and the signal was passed through a final $1 \times 1$ convolution before it is finally upsampled. The network achieved an mIoU of 81.3 on Cityscapes, which again is a considerable improvement over the previous version.

The latest version, **Deeplabv3+**, extends the original network by replacing the ResNet backend with a modified version of the Xception network and additionally increases the performance by improving the object boundaries (Chen et al., 2018). This was done by implementing an encoder-decoder structure instead of directly upsampling the class scores to the desired image size with bilinear interpolation. For the decoder, the authors introduced a stepwise upsampling followed by a skip-connection from an earlier layer of the same resolution, followed by two convolutions. The idea of the skip connection is that it helps to recover the position of an object from an earlier low-level layer. Often, this type of structure is referred to or compared with U-net by Ronneberger et al. (2015). The network achieved an mIoU of 82.1 on Cityscapes.

The current state of the art in semantic segmentation on Cityscapes is **HRNet** (Sun et al., 2019). Many researchers have realised that a pure encoder-decoder structure has its limits due to inaccuracies in the last prediction layers because of the downsampling process in the encoder. To

overcome this problem, many models use skip connections between earlier layers of the encoder to later layers of the decoder (Chen et al., 2018; Ronneberger et al., 2015). The authors of HRNet argue that the interconnection between low and high level layers is a poor architectural choice, as it only leads to rich low-resolution or poor high-resolution representations in the upsampling process. Their proposed network (see Figure 3.4) does not follow an encoder-decoder structure, but maintains the feature maps at the original image size from the beginning to the end of the network. However, as it can be seen in Figure 3.4, the network implements downsampling and upsampling in parallel with the high-resolution feature maps to produce high-level features at a lower resolution. These lower-resolution feature maps are then upsampled in every other step and passed on to all higher-resolution branches. This network achieved an mIoU of 84.5% in Cityscapes.

## 3.2 Semantic Segmentation (3D)

Contrary to the possible assumption that processing 3D spatial data is simply the extension of 2D spatial data, it is actually more complicated. Hence, the 3D semantic segmentation is described here separately. While 2D convolution has become a standard for image processing because images are very dense, there is no obvious architectural choice for ANNs in 3D. There are many factors to consider, such as efficiency, performance and the data structure provided. 3D convolutions may be the first choice for processing 3D data, but they are probably the least efficient. Also, unlike image-based processing, 3D data is often unstructured, can come in different representations and varying densities. In this section some standard procedures are explained which will be used in the further work. For a detailed overview of current 3D semantic segmentation networks, see the work by (Zhang et al., 2019b).

Similar to 2D semantic segmentation, the origin lies in the segmentation of 3D point clouds with, for example, edge-, region- ,model-, cluster- or graph-based methods (Nguyen and Le, 2013; Grilli et al., 2017; Xie et al., 2020b). For semantic segmentation in 3D, many regular machine learning methods have been proposed that are not based on deep learning, such as SVMs (Zhang et al., 2013; Li et al., 2016; Mallet et al., 2008), Boosting (Wang et al., 2015; Lodha et al., 2007) or Random Forests (Chehata et al., 2009). The general procedure for these machine learning based methods was described by Weinmann et al. (2015), which consists of (i) neighbourhood selection, (ii) feature extraction, (iii) feature selection and (iv) semantic segmentation. As Niemeyer et al. (2014) argue, many of these models have not taken into account the context of each point and processed them independently, leading to inhomogeneous results. Conditional random field-based classifiers helped to overcome this problem by smoothing neighbouring predictions (Vosselman et al., 2017; Niemeyer et al., 2014, 2012; Schmidt et al., 2012; Lim and Suter, 2009). In contrast, deep learning based methods in 2D avoid this problem by using stacked convolutions in order to increase their receptive field and take neighbouring pixels into account to make a prediction. Araujo et al. (2019) showed that, depending on the backbone, the receptive field for the final prediction layer can be as small as $195 \times 195$ pixels with AlexNetv2 up until $3039 \times 3039$ pixels with InceptionResNetV2, which covers the entire input image. Deep learning based solutions for 3D semantic segmentations heavily depend on the choice of point cloud representation. In general, they can be divided into methods that use grid-like structures, such as projection-, multi-view- or voxel-based methods, and methods that work directly on the raw point cloud (Zhang et al., 2019b). Next, each of these types is will be discussed individually.

### Projection-Based Methods

Point clouds can be discretized into many different structures, which allows the application of more classical approaches. For example, projection-based methods, also called multi-view-based

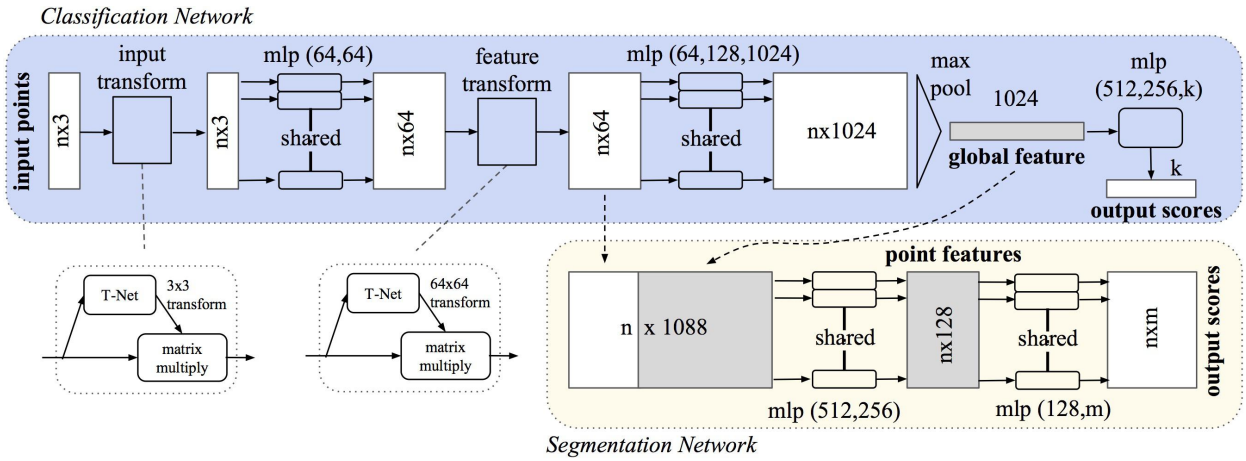



*Figure 3.5: Schematic overview of point net. Image Source (Qi et al., 2017a)*

methods (Xie et al., 2020b), were used in early attempts to solve semantic segmentation in 3D by applying 2D convolutional networks to projected 3D data. Wu et al. (2018) and Boulch et al. (2018) projected a point cloud into 2.5D images using Equation 2.4 with a virtual camera. However, this leads to occlusions, which have to be treated e.g. by ray tracing, which can be computationally expensive. Additionally, the choice of intrinsic and extrinsic camera parameters will introduce many new hyperparameters. As described by Wang et al. (2019), object detection in images can be significantly worse in 3D because 2D kernels are not invariant to object size and depth. For example, the same objects may appear in different sizes due to their distance to the camera center, making distant objects smaller and more difficult to detect. Furthermore, due to the projection, neighbouring pixels in 2D may not correspond to neighbours in 3D, which can lead to a wrong context and introduce unnecessary or even wrong information.

In this work, scanstrips are used as a representation for point clouds. There are many works that perform semantic segmentation specifically in scanstrips using 2D CNNs (Wu et al., 2018; Wang et al., 2018b; Wu et al., 2019; Behley et al., 2019; Milioto et al., 2019; Qiu et al., 2019; Biasutti et al., 2019b,a). Except for the work of Qiu et al. (2019), all the approaches use spherical projections of the Velodyne HDL-64e LiDAR from the KITTI dataset by (Geiger et al., 2013), therefore, they mainly differ in the network architecture and the selected features.

### Voxel-Based Methods

Another possibility is the discretization of 3D data into a 3D grid, often called voxelization. This representation is used in this work for shape completion of occluded surface points on objects. Voxelization preserves the local relationship of points and is much easier to implement, because 2D architectures of neural networks only need to be extended by one dimension, which can be achieved by expanding the sum in Equation 2.34 by a third dimension. However, the discretization step leads to a loss of resolution and the convolution is very inefficient, because most of the volume is typically empty, leading to both memory inefficiency and wasted computation. Popular voxel-based networks are VoxNet by Maturana and Scherer (2015), SEGCloud by Tchapmi et al. (2017) and PointGrid by Le and Duan (2018). Some methods like OctNet by Riegler et al. (2017a) tried to avoid the problem by introducing tree-like structures. In OctNet the voxel size is adaptive, based on the local density of the grid.

### Raw Point Clouds

The problem with point clouds is that, unlike grid structures, they are provided in no particular order. The first ANN to perform classification and semantic segmentation on raw point clouds was PointNet by Qi et al. (2017a). As shown in Figure 3.5, they encoded each point separately into a higher order feature vector and reduced it to a global vector of fixed size using max pooling. Max pooling is invariant to the order and length of the list of points, so it does not affect the resulting fixed-size feature vector. The global vector can then be passed to a few dense layers that output the class distribution. In the case of semantic segmentation, the global feature vector is concatenated to each point in the latent space. Afterwards, the resulting list of points is transferred to a few dense layers, which finally predict a point-wise class distribution. However, the problem with PointNet is that it only has a receptive field of one before it is reduced by the maximum pooling layer. Hence, local characteristics cannot be taken into account to classify a particular point. To illustrate the problem, this is equivalent to asking for the class of a certain point, while having only the information about the room size in which the point is located. For this reason, PoinNet++ was developed by Qi et al. (2017c), which uses many local PointNets to create a hierarchical structure, leading to a receptive field larger than one. This is done by grouping the 3D points into spheres. Each sphere is encoded into a local feature vector using PointNet with max-pooling. This feature vector is then assigned to the points and passed to the next layer of PointNet++, which then performs a query with larger spheres and so on. This method is used in this work to create a feature representation of objects for shape completion.

A much more recent development is KPConv by Thomas et al. (2019). Here, the 3D points are treated as discrete samples in a continuous convolution. They created a network that performs the convolution directly in 3D space without discretizing the points into a grid-like structure. The spatial relationship is taken into account by weighting each point by its distance from the kernel center. In contrast to a grid, the kernel center is the central query point. The neighborhood of this point is defined by all points within a spherical neighborhood of fixed size. The resulting feature vector is then stored directly at this point. Successive KPConv layers can access the feature point in the same way by querying surrounding feature points, resulting in a typical hierarchical neural network structure. Until today, KPConv is the best performing method for most 3D benchmarks.

Other notable works for 3D semantic segmentation include PointCNN (Li et al., 2018b), SO-Net (Li et al., 2018a) and RSNet (Huang et al., 2018). PointCNN emulates convolutional layers by learning how to transform points in a local coordinate system to apply a traditional convolutional operation directly to the points. To do this, they use an MLP that learns how to order the points before applying the kernel. On the other hand, SO-Net uses self-organizing maps in order to learn a permutation invariant representation of the point distribution. Finally, the authors of RSNet developed a slice pooling layer that projects unordered point features into a regular order to which an RNN is then applied. After that the point features are unpooled in order to assign them to each point in 3D for further processing.

In general, all of these works are important for this thesis because they describe how 3D point clouds can be segmented semantically and what types of representations are used, all of which have different advantages and disadvantages. In particular, PointNet has shown how to deal with unstructured data. This method is adapted in this work to deal with multi-view images, which in this sense have similar characteristics.

## 3.3 Semi-Supervised Learning

This subsection will introduce state-of-the-art semi-supervised learning methods that are related to this thesis as well as topics which are only weakly related to semi-supervised learning but are intersecting with this thesis. Semi-supervised learning describes a class of algorithms that learn from a combination of unlabelled and labelled data. The unlabelled data is often obtained from the same source as the labelled data. The reason for doing this is to increase model performance in terms of precision and generalisation since often the amount of labelled data alone is not sufficient. Semi-supervised learning can help as long as the distribution of unlabelled examples extends the labelled information and does not mislead the model. To be more precise, according to Chapelle et al. (2010) the following assumptions have to hold in order for semi-supervised learning to work:

- **The Continuity Assumption** also referred to as smoothness assumption describes that if the input data $x_1$,$x_2$ are close to each other in a high density region, the corresponding output $y_1$,$y_2$ should also be close to each other.
- **The Cluster Assumption** can be seen as an extension of the previous assumption. The assumption holds if points that form a cluster also belong to the same class. In this case, it is usually beneficial to use semi-supervised learning to help the model find the decision boundary of each cluster more accurately.
- **The Manifold Assumption** as described by Chapelle et al., "the (high-dimensional) data lie(s) (roughly) on a low-dimensional manifold" (Chapelle et al., 2010, p.6). The problem described here relates to the curse of dimensionality. When high-dimensional data is generated by a process with few degrees of freedom, it lies on a lower-dimensional manifold. If data points on the same manifold have the same label, the class assignment of unlabelled points can be made from labelled points on the same manifold

The taxonomy of semi-supervised learning describes many different types of approaches, which can be divided into inductive and transductive methods (Van Engelen and Hoos, 2020). In inductive methods, a classifier is trained to map from $X$ to $Y$. In contrast, transductive methods do not build a predictive model, here the entire dataset is used to simply derive labels for the given unlabelled data. Closer to the topic of this thesis are inductive models called "wrapper methods" and "unsupervised preprocessing" which are presented below.

**Wrapper methods** are "wrapped" around a supervised base classifier of any type in order to utilize unlabelled data. According to (Chapelle et al., 2010, P. 3), one of the earliest semi-supervised learning methods is **self-training**, which was proposed by Scudder (1965). The algorithm works by first training a model on the supervised data. In the next iteration, the unlabelled data is annotated by the model from the previous step. Next, the training is restarted on the entire dataset (true labels and the own predictions). These steps are repeated until the best model is found. Due to the nature of this algorithm, self-training has been applied to many different artificial neural networks for classification (Xie et al., 2020a; Zoph et al., 2020) and very recently to semantic segmentation (Zou et al., 2018; Zhu et al., 2020; Feng et al., 2020). Even though semantic segmentation using deep neural networks is a relatively new topic, the basic principle of self-training stayed the same. For example, Zhu et al. (2020) used self-training with Deeplabv3+ by first training a "teacher" network on 5k finely annotated ground truth images from Cityscapes, and then training a "student" network jointly on the ground truth and the teacher-generated pseudo-labels on another 20k unlabelled Cityscapes images. A very similar approach was shown by Zoph et al. (2020) on different datasets and by Xie et al. (2020a) for classification. Closely related to self-training is **pseudo-labelling**. This approach was introduced by Lee et al. (2013) using neural networks. The algorithm differs from self-training in the way that the network is not re-trained entirely in each iteration. Instead, unlabelled data points are pseudo-labelled throughout the training process. Similar to self-training

pseudo labelling has been used for classification (Wu and Prasad, 2017) and semantic segmentation tasks (Zou et al., 2020; Yao et al., 2020; Chen et al., 2020b).

**Unsupervised Preprocessing** usually describes a two-step approach (Van Engelen and Hoos, 2020, p. 393). As the name "Unsupervised Preprocessing" suggests, the first step involves unsupervised preprocessing followed by a semi-supervised or supervised learning step (Van Engelen and Hoos, 2020, p. 393). There are some different types of preprocessing, such as clustering, but the ones closest to the topic of this thesis are feature extraction and pretraining (Goodfellow et al., 2016, p. 517 ff.), which are discussed below. A very prominent example is the autoencoder (Goodfellow et al., 2016, p. 346f.). The autoencoder is a network that tries to minimize the distance between its prediction and the input. To avoid trivial solutions, the autoencoder typically consists of an encoder stage and a decoder stage, much like the semantic segmentation networks discussed earlier. The centre of the network, often referred to as the "bottleneck", has a much lower dimension than the input and output, which forces the network to project the data onto a low-dimensional space to find a latent representation of the data. It is worth noting that this idea is closely related to the Manifold Assumption described earlier. Once the network is trained the encoder part can be used to create a latent representation of any typical input data, which can be used in the second training-step for supervised training using a much smaller neural network. It should be emphasized that for these methods to succeed, the other two assumptions must also hold in order to map unlabelled data from latent representations to the correct class label. Well-known autoencoders are the denoising autoencoder by Vincent et al. (2008), contractive autoencoder by Rifai et al. (2011) or generative models such as variational autoencoders by Kingma and Welling (2014).

**Pretraining** is related to feature extraction (Van Engelen and Hoos, 2020, p. 396) and **transfer learning** and also domain adaptation (Goodfellow et al., 2016, p. 525). In (greedy layerwise) unsupervised pretraining (Goodfellow et al., 2016, p. 519) a neural network is trained on its objective, while gradually new layers are added and the others frozen (Goodfellow et al., 2016, p.314,p.520). Historically, this has been done because some networks where to big or unstable to be trained end-to-end, see Bengio et al. (2006) and (Goodfellow et al., 2016, p. 519). Another form of pretraining that is more closely related to transfer learning (Goodfellow et al., 2016, p. 315) is when a network is pretrained in a supervised manner on one dataset and later fine-tuned on similar dataset.

#### Transfer Learning and Domain Adaptation

Transfer learning and domain adaptation assumes that knowledge from one domain (source) can be helpful in another domain (target) (Goodfellow et al., 2016, p. 526-527). In general, the goal is to improve the classifier in the target domain by transferring knowledge from the source domain. Like semi-supervised learning, it is also often done to reduce the cost of labelling. In **transfer learning**, this can be achieved by transferring the weights of a network that has been fully trained (pretrained) on the source dataset to the target dataset, where they are fine-tuned. Often the target dataset is relatively small, so fine-tuning is done by freezing the first layers and adding a regularization loss to avoid overfitting. The initial weights serve as a starting point to find a better global optimum than with a randomly initialised network. As Yosinski et al. (2014) have shown, the first layer often extracts features such as edges, corners or shapes that occur in many different image domains, thus laying the foundation for transfer learning. It should be noted that the target domain does not necessarily have to have the same classes as the source domain. Also, transfer learning is not limited to semi-supervised learning, as it can also be helpful if the target dataset is fully annotated. Extreme cases of transfer-learning are **one-shot** and **zero-shot learning** where only one or no examples in the target domain are available for fine-tuning (Goodfellow et al., 2016, p. 529). Nowadays, transfer learning is ubiquitous. As already described in Section 3.1, most semantic segmentation networks use interchangeable backends that have almost always been

pretrained on a classification dataset such as ImageNet (Krizhevsky et al., 2012). Other notable publications are Huang et al. (2013); Long et al. (2016); Luo et al. (2017); Zhu et al. (2016); George et al. (2017); Chang et al. (2018).

**Domain adaptation** is a special case of transfer learning. It is done when the classes of the target and source domains as well as the sensor type coincide, but the input domain is different (Goodfellow et al., 2016, p. 527). In this case, the goal is to adapt the feature detectors to the target domain as such, so that the network predicts the correct classes in the new domain." Depending on whether there are annotations in the target domain, it is referred to as supervised or unsupervised Domain Adaptation (Wang and Deng, 2018, p.3). Many existing deep learning methods aim to solve this problem by training a discriminator that learns to differentiate between the extracted features of the source and target domains. The classifier, on the other hand, learns to fool the discriminator and hence closes the domain gap (Tzeng et al., 2017; Hoffman et al., 2018; Cao et al., 2018).

Transfer learning is an important part of this work. In this thesis, a network will be trained on only few labels to semantically segment 3D point clouds. With transfer learning, a network would be pretrained on 3D point clouds and then finetuned on the target dataset. However, unlike for 2D images, there are only few pretrained networks available for 3D point clouds. Therefore, it is shown that it is possible to transfer features generated by a pretrained network on 2D images to 3D point clouds to achieve better results in 3D semantic segmentation than using supervised training directly. Domain adaptation also plays an important role in this process, as the pretrained network may suffer from a domain gap that needs to be addressed.

### Self-Supervised Learning

The term self-supervised learning describes a class of algorithms that generate a supervisory signal from the data itself. Self-supervision is very close to semi-supervised learning but should not be confused with self-training, which almost always solves a task labelled by a pretrained teacher. In self-supervision, the reference is generated automatically, rather than by a teacher which was trained on a reference dataset. Often self-supervision is divided into two steps (Jing and Tian, 2019). First, a base learner must solve a pretext task, which, as the name suggests, serves as an auxiliary problem for representation learning to help solve the actual task in the second step. The second step is called the downstream task, where the model from the pretext is trained to solve the actual problem. According to Jing and Tian (2019), the downstream task often uses annotated data. However in some cases the downstream task can be done without using any annotations. For example, Sermanet et al. (2018) trained a robot to imitate videos without any direct supervision. Here, the pretext task was to create representations of video frames by predicting the next frame using a triplet loss (Chechik et al., 2010). The downstream task was to learn a policy using reinforcement learning that attempted to create similar representations by having a robot imitate the poses similar to the video. Another example for self-supervision by Vondrick et al. (2018) shows how to track objects in videos (downstream task) by learning how to correctly copy color from one frame into the following grayscale frame (pretext task). Image colorization can be trained in an unsupervised way by removing the color information and training a model to predict color from grayscale images. This is very similar to inpainting images, where parts of the image are removed so that a neural network must learn to complete the missing part of the image (Yeh et al., 2017; Yu et al., 2018; Demir and Ünal, 2018). Something related was done by Noroozi and Favaro (2016) by training a network to solve puzzles from images to learn representations of the images. Similarly, Doersch et al. (2015a) trained a neural network to predict the relative position between two random patches from one image. Numerous other pretext tasks have been used in computer vision. Some of them try to learn representations based on the distortion of the images (Dosovitskiy et al., 2015),

or rotating images (Doersch et al., 2015b). Both created surrogate classes by distorting images and trained a neural network to classify the real images within the distorted images.

In this work, self-supervision is used to complete 3D shapes. The goal is to predict a complete shape from an incomplete input. The supervision signal for training the network is automatically generated from areas of the object with high point density. The difficulty to overcome here is that often complete shapes are not available, so the task is solved completely unsupervised.

## 3.4 Conditional Generative Adversarial Networks

The Conditional Generative Adversarial Network (CGAN) is a special type of GAN that learns to produce an output based on a given input (condition) (Mirza and Osindero, 2014). In a normal GAN, the generator learns a mapping from a distribution $z$ to a distribution $y$, i.e., $y = G(z)$. In the conditional version, a condition $x$ is introduced to the generator as such $y = G(x,z)$. The discriminator is trained in almost the same way as a GAN, but takes into account tuples of the condition and the target distribution $D(x,y)$. Because of this property, a CGAN can be viewed as a supervised version of a GAN, since it is often (but not always) necessary to use fixed pairs of input and output samples. As shown in the Auxiliary Classifier GAN (ACGAN) by Odena et al. (2017), this can be improved by providing the discriminator with additional class labels that force the generator to respond appropriately to the condition and not to ignore it.

Of particular interest for this thesis are the networks pix2pix by Isola et al. (2017) and the improved version pix2pixhd by Wang et al. (2018a), which is able to predict high-resolution image-to-image mappings. This means that these networks can learn to map between (paired) images, for example, to predict realistic-looking images based only on labelled images or satellite images based on maps. Similar to semantic segmentation, the generator networks follows the general encoder-decoder architecture such as the U-net (Isola et al., 2017) or using residual connections (Wang et al., 2018a). However, due to the discriminator network, the generator can learn to produce very realistic looking images.

**Multimodal Image-to-Image Translation** defines the process of mapping from one input to many outcomes. For example, Zhu et al. (2017) uses an additional latent space vector as input to the generator, which helps to encode the possible distribution of results for an input image. During test time, they can change the latent vector to produce different looking results for the same input image. On the other hand, Wang et al. (2018a) used semantic and instance labels to train an encoder for each object instance in images. The encodings were passed to the generator during training. By exchanging the encodings during inference, they were able to manipulate the appearance of each object in the generated images. Multimodal image-to-image translation is also used in this work. Here, a network that has a similar structure to pix2pixhd is trained to map point clouds to realistic looking images. It is shown that it is possible to control the appearance of the synthetic images and thus map a point cloud to multiple possible outcomes.

## 3.5 Multi-View Fusion, Prediction and Labeling

Another central topic of this work is multi-view label transfer and semantic segmentation. The term multi-view describes the observation of objects or a scene in multiple views, i.e. from different camera positions. A well-known example from photogrammetry is 3D reconstruction from multiple images using structure from motion or a calibrated stereo camera. Apart from that, multi-view data has a very wide field of application and there are many publications and approaches to solve different problems. For example, in machine learning, multiple observations are used to improve

the prediction quality for classification (Su et al., 2015), semantic segmentation (Xiao and Quan, 2009), or object recognition (Nassar et al., 2019).

This work is related most closely to semantic segmentation and classification in multiple views with CNNs. Several approaches have already been presented to tackle classification in multiple views. They mostly differ in the strategy of how the information is fused between the multiple views. Seeland and Mäder (2021) showed a comprehensive overview of feature fusion and presented three general strategies. They categorized the fusion methods into **early-**, **late-**, and **score-fusion**. In early fusion, feature maps from multiple views are stacked and passed to a common network. Late fusion, on the other hand, has multiple branches that are aggregated just before or in the final classification layer. Finally, score fusion uses voting on all individual predictions from multiple views. They are aggregated by summing the softmax predictions to produce the final vote. In general, most works fuse multiple views using functions such as the maximum of the features (Su et al., 2015), the weighted sum (Feng et al., 2018), the sum (Dolata et al., 2017), the product (Do et al., 2017), an RNN (Lee et al., 2018), a learned transformation (Wang et al., 2015) or feature concatenation (Lin and Kumar, 2018; Setio et al., 2016; Barbosa et al., 2020; Geras et al., 2017).

In addition to classification, some approaches have been proposed in the past that use **multi-view consistency** to improve predictions in general. For example, Floros and Leibe (2012) used a Conditional Random Field (CRF) to enforce temporal consistency between video frames to perform semantic segmentation. Another CRF-based method was proposed by Hermans et al. (2014). They reconstructed a point cloud using sequences of classified RGB-D images and refined the predictions using a CRF. For consistency across multiple views, Ma et al. (2017) presented a deep learning based approach to make predictions consistent across multiple views. To do this, they warped the feature maps of multi-view RGB-D images into a common reference frame. To improve the predictions in an unsupervised manner, Zhou et al. (2018) forced 3D keypoint predictions to be consistent in space. Finally, a recurrent neural network-based approach for 3D mesh segmentation was presented by Le et al. (2017).

**Label transfer** describes the transfer of labels from one domain to another using a mapping function, such as a the camera matrix that can map 3D point labels to 2D images. Three methods are close to the topic of this work. First, Xie et al. (2016) uses annotated point clouds to transfer labels from 3D to 2D. As it is relatively easy to capture new images in the same reference frame as the annotated point cloud, for example using a mobile mapping system, they can generate an arbitrary number of annotated images. The reverse direction of label transfer was performed by Zhang et al. (2018) and Peters and Brenner (2019). They used semantically segmented 2D images and projected them into point clouds. However, Zhang et al. (2018) did this specifically for static scenes using a terrestrial laser scanner, due to the problem of dynamic occlusions. This means that errors in the label transfer due to moving objects is not considered in the work by Zhang et al. (2018). Peters and Brenner (2019), on the other hand, tried to solve this type of problem by introducing two features to detect false mappings of 2D pixels to 3D points. It was shown that using these features can help to correct incorrect labels in 3D. This thesis extends the work of Peters and Brenner (2019) by learning to transfer labels from images to 3D point clouds in a end-to-end fashion, resulting in far fewer errors and much better semantically segmented point clouds.

In this thesis, several network architectures are built using 2D multi-view image observations in combination with a PointNet-like architecture to estimate a consistent class label between all corresponding multi-view image patches. As Peters et al. (2020) have shown, it is possible to re-train or fine-tune segmentation networks using the multi-view predictions, see Fig. 3.6. In this Figure each row shows an example from their work. The second column shows that HRNets initial predictions are very poor for some classes. The third column shows that it was possible to recover

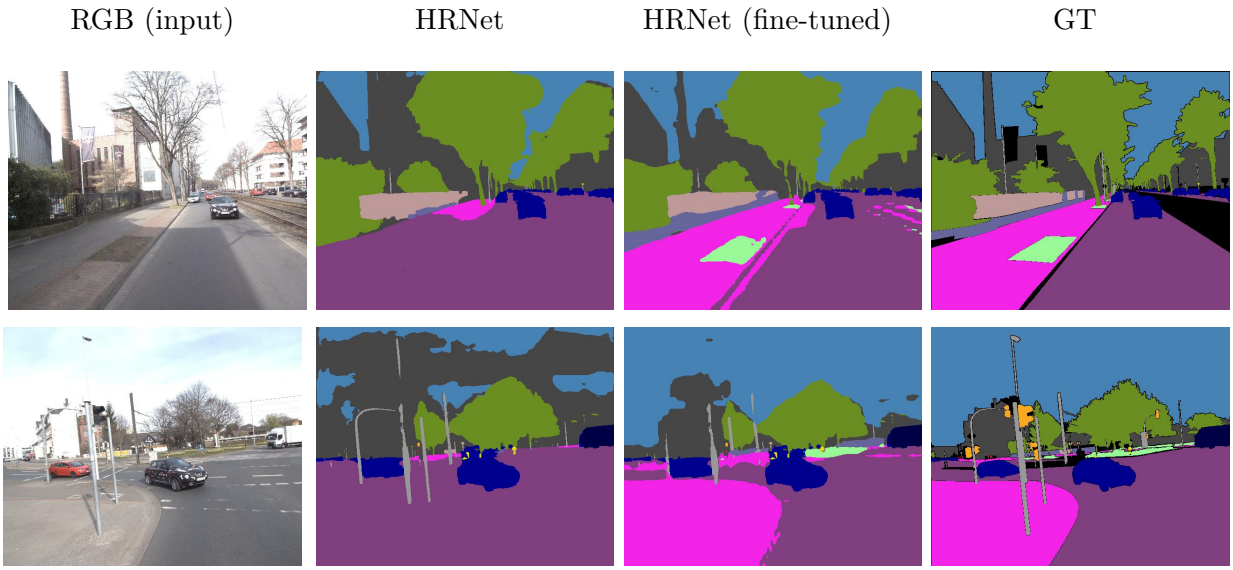



*Figure 3.6: Examples for successfully corrected predictions after retraining of HRNet as presented by Peters et al. (2020). The figure shows, from left to right, the input image, the uncorrected prediction, the corrected prediction, and the manually labelled ground truth.*

the lost classes by fine-tuning the network. Fine-tuning was done by first correcting the initial predictions of HRNet with a multi-view network and then retrain HRNet using the corrected predictions.

The assignments for the multi-view images were made by projecting a 3D point onto all associated images. This creates a list of the corresponding image pixels. It was then assumed that all associated 2D pixels belong to the same object. The multi-view network was trained to predict a single class based on the pixels of the multi-view images. The problem is that related pixels very rarely belong to the same class. For example, moving objects, occlusions, or calibration errors can cause the wrong pixels to be assigned to a 3D point. In this work, this problem is addressed by introducing a multi-view network that receives a list of multi-view image pixels and outputs a prediction for each multi-view observation instead of a single prediction for all of them. This leads to better predictions on multi-view images and eventually to better results when a network is re-trained using the corrected predictions.

## 3.6 Shape Completion

3D shape completion describes the completion of 3D objects that have only been partially observed. It is a long-standing topic in computer vision and computer graphics (Pauly et al., 2005; Dai et al., 2017). Classical approaches were often based on deforming a shape template to fit to the data (Kraevoy and Sheffer, 2005) or fitting a generic deformable surface model (Fedkiw et al., 2001). For regular, human made objects, many authors have also used symmetry to fill in missing parts by finding ways to copy, extrude or mirror existing ones (Mitra et al., 2006; Podolak et al., 2006; Pauly et al., 2008; Mitra et al., 2013; Sipiran et al., 2014; Sung et al., 2015).

Shape completion is also related to inpainting, which refers to completing partially observed images. This can be achieved by for example pasting in a content that can be copied from the image itself (Criminisi et al., 2004) or synthesised from a collection of training images (Yang et al., 2017).

More recently, learning based approaches are favoured for shape completion. Many of them operate in a fully supervised setting, assuming that paired training data with incomplete and complete versions of the same shapes are available (Yuan et al., 2018; Firman et al., 2016; Han et al., 2017; Rezende et al., 2016; Riegler et al., 2017b). Supervised learning has also been combined with symmetry for the task of face Inpainting (Zhang et al., 2019a). While such strong supervision simplifies the task from a machine learning point of view, a big effort is required to assemble a sufficiently large training set. More importantly, it requires adapting the system to the specific sensor characteristics and environment of the training data, which is difficult to translate to other scenarios. Therefore, some works examine weakly supervised settings where only a small portion of the data is labelled (Chen et al., 2019). For example, Stutz and Geiger (2018) trained a variational auto-encoder (Kingma and Welling, 2014) for shape completion on synthetic 3D shapes in voxel representation. To achieve shape completion on the target dataset, they then fine-tuned the encoder using a small set of real 3D point cloud data extracted from KITTI. An issue with synthetic training data is that it is difficult to simulate realistic scanning conditions, with, for example, reflections and boundary effects. Another recent trend is to exploit unpaired input and output shapes, which has been done by Lu and Dubbelman (2020) and Chen et al. (2020a). Other notable approaches that were not explicitly designed for shape completion are (Yang et al., 2018) and (Sharma et al., 2016), who created auto-encoders for 3D data. However, none of these works is capable of completely unsupervised learning for shape completion, i.e., without ground truth, which is addressed in Section 5.2 of this thesis.

# 4 Multi-View Label Transfer and Correction

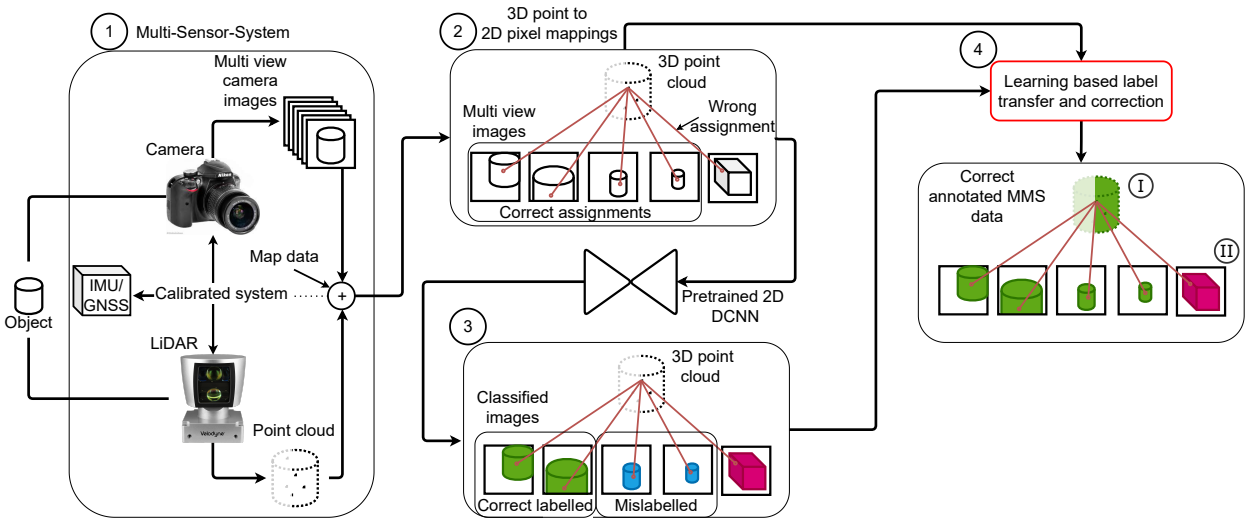



*Figure 4.1: The diagram shows the general procedure of label transfer and correction process, described in this chapter. (1) First the data is acquired using a fully calibrated MMS. (2) Secondly all matches between 3D points and 2D pixels are calculated by projecting the 3D points into the corresponding images. (3) These images are pixelwise classified by a pretrained DCNN, where different colors indicate different predicted classes and different shapes represent different object classes. Note the (magenta) cube in boxes 3 and 4 indicates a wrong assignment between pixel and 3D points. (4.I) By learning how to transfer the labels from 2D to 3D, the point cloud is correctly labelled. (4.II) By observing the predictions from several views in 2D, it is learned how to correct the initial predictions.*

The aim of this work is to minimise the annotation effort of 2D and 3D data by exploiting the geometric correspondence in a fully calibrated mobile mapping system with IMU/GNSS, LiDAR and cameras. Figure 4.1 shows the general procedure in four steps, which are explained in more detail in this section.

In a fully calibrated MMS, both the extrinsic pose of each image and the intrinsic camera parameters are known. Furthermore, all camera poses and the point cloud coordinates are known in the same global world coordinate system, see Figure 4.1 (box 1). This prerequisite makes it possible to find the matching 2D image pixels for the 3D point cloud points, see Figure 4.1 (box 2). By grouping all pixels for each 3D point, it is also possible to identify all images showing the same 3D point of an object, see Figure 4.1 (box 2). Here, the point cloud of a cylinder is shown, where the red lines indicate the corresponding pixels in each image, showing all images of the cylinder. However, due to various errors, such as wrong assignments due to occlusions or calibration errors, this process may establish incorrect matches between 3D points and pixels in a 2D image, which is represented by the image of the cube in Figure 4.1 (box 2).

The arrow between boxes 2 and 3 in Figure 4.1 shows the use of a pretrained DCNN that makes a pixel-wise classification of all multi-view images, leading to box 3. This is the starting point for the following methods and procedures in this chapter. The colors in box 3 indicate the classes predicted by the DCNN. Here, the pixels of two cylinders were misclassified (blue) and two were

classified correctly (green). This chapter addresses the following problems, which are indicated by box 4 (I) and (II):

The first problem is (box 4.I): **Is it possible to assign the correct labels to 3D point clouds from multi-view images that contain various wrong assignments or wrong predictions?** Here, the geometric correspondence between LiDAR and camera can be used to transfer labels classified by a pretrained DCNN into the 3D domain to annotate large amounts of 3D data (Peters and Brenner, 2019). However, the direct mapping from 2D to 3D leads to incorrectly annotated data in 3D (label noise). These errors are mainly due to calibration and classification errors, incompatible label policy and occlusions. This chapter therefore addresses the reasons for these errors, how they can be dealt with and how they are eventually corrected.

The second problem is (box 4.II): **Is it possible to correct wrong classifications in 2D images by viewing the same object multiple times from different perspectives?** Here, the problem is addressed that the pretrained DCNN performs worse on MMS images than on the publicly available dataset on which it was trained (Peters et al., 2020). This is mainly due to different camera models, different camera settings, new (unknown) viewpoints and a new environment. This chapter shows that the correspondences between multi-view images (step 2) can also be used to bridge the domain gap for the pretrained DCNN between the publicly available dataset and the MMS images by forcing the 2D predictions of multiple views of the same objects to be more consistent in 3D.

In both cases, the aim is to reduce the need for annotated data in 3D point clouds or 2D images by introducing semi-supervised learning schemes that learn how to transfer information between both domains. In Section 4.1, steps 1,2 and 3 are explained in detail as they are the same for method I and II.

## 4.1 2D to 3D Label Transfer

Since all individual laser measurements and all captured images are time-stamped, the complete geometry can be reconstructed using the IMU/GNSS data, so that each 3D point can be projected into each image using the outer and inner orientation and lens distortion terms. As depicted in Figure 4.1 (box 2) the general connection between a 2D pixel $u_k$,$v_k$ in an Image $I_k$ with $k \in \{1,\ldots,N\}$ and a 3D point $p_j = (X_j,Y_j,Z_j)$ is established by projecting each 3D point $j \in \{1,\ldots,M\}$ into every image $I_k$ by using Equation 2.4. This results in a one to many connection, which means that one 3D point can be possibly projected into $N$ images.

With the help of this mapping method it is possible to assign a list $l_j = \{I_1(u_1,v_1),\ldots,I_k(u_k,v_k)\}$ of 2D pixel coordinates to each 3D point $p_j$. Note that the length of the list depends on the number of available images and can vary from zero to $N$. Also, the list contains many incorrect mappings that lead to **errors** in the label transfer process, such as **regular, self-** and **dynamic occlusions**, **label policy** errors and **calibration** errors. In the following, the error causes are briefly presented. The severity of the errors and their solution are examined in the experimental part.

### 4.1.1 Regular and Self-Occlusions

Since laser beams and the image beams do not coincide, occlusions are very common, resulting in 3D points projected into images which are occluded by other objects. The assignment of RGB values from images to 3D point clouds can be used to show this effect. Figure 4.2 shows two objects that are colored incorrectly due to occlusion. The examples show a facade that is partly colored

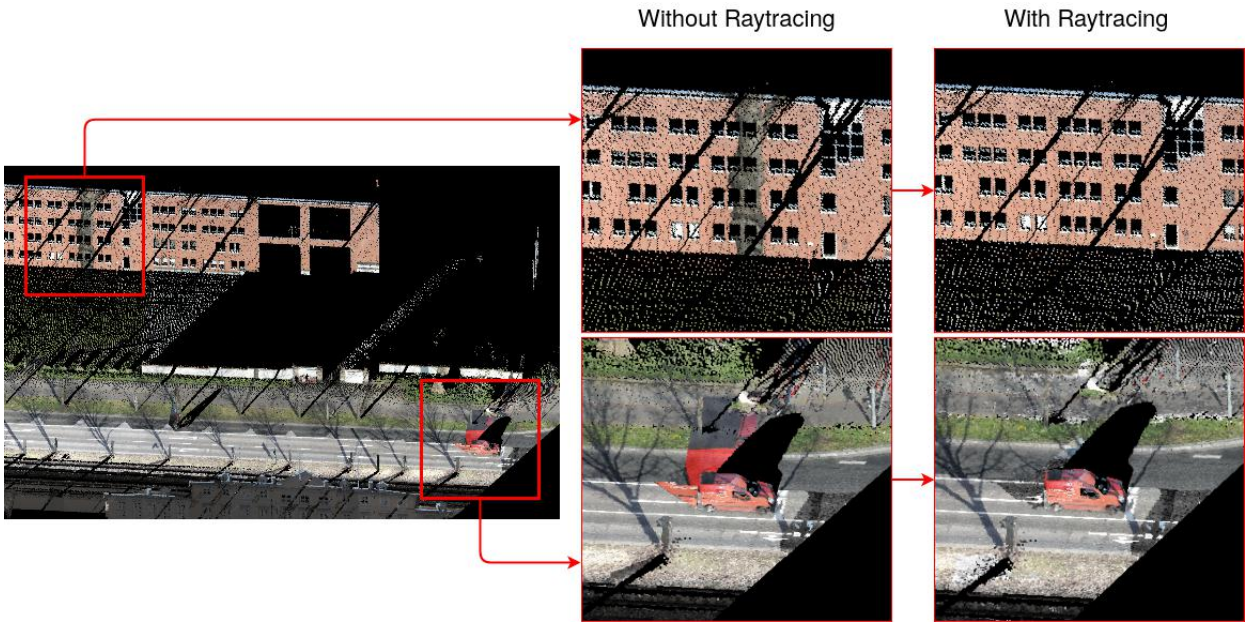
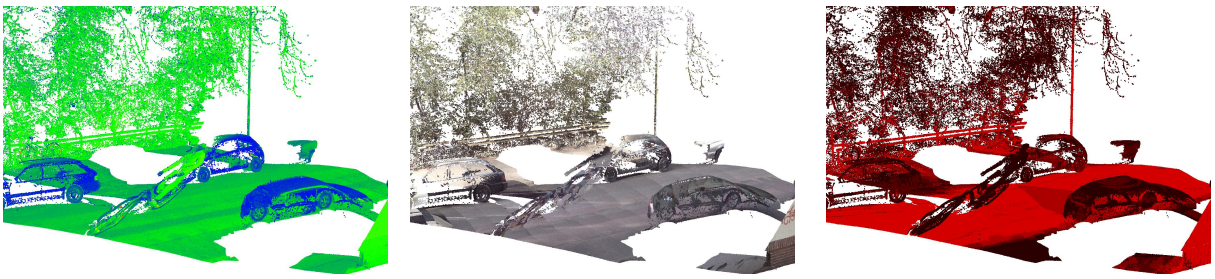



*Figure 4.2: A point cloud with examples for regular and self-occlusions. The tree and car are once colored using the nearest RGB value and once colored using ray tracing.*

as tree and a car colors that leaks into the environment, which means that the street and sidewalk are colored red.

To prevent this, a full ray tracing is used[1] as described in 2.1.2.2. The results of the ray tracing process is depicted in Figure 4.2 (boxes on the right). It shows that points on the facade were correctly detected as being occluded so that no tree pixels were assigned to them. The same can be seen in the example with the car, where neighbouring objects were successfully detected as occluded so they received their color information from different views in which they were not occluded.

### 4.1.2 Dynamic Occlusions



*(a)* Point cloud colored by reflectance

*(b)* Point cloud colored by assigned RGB value with ray tracing

*(c)* Point cloud colored by campaign count $\xi$

*Figure 4.3: A point cloud colored according to the point reflection (left), the RGB color (middle) and campaign count $\xi$ (right). Although the the color was assigned using ray tracing, the cyclist in the center of the images is incorrectly colored.*

Even if occlusions are detected using ray tracing, the mapping procedure only works for static objects, as shown in the following example. Figure 4.3 shows three images of the same scene; the

[1] The ray tracing algorithm with occlusion culling was provided by Prof. Claus Brenner

left image 4.3a is colored by the point reflection and the middle image 4.3b is colored with the corresponding RGB value from the image with the smallest distance between the 3D points and the camera centre. The scene shows a cyclist in the centre of each image surrounded by three parked cars. Because the cyclist was in motion when the laser scanner captured him, he appears stretched in the direction of travel. Even more striking is that the assignment of the correct RGB values for the cyclist has completely failed, as he does not appear in the corresponding camera images. Therefore, the cyclist appears in Fig. 4.3b in the same color as the neighboring road and the car. This example shows a severe error case where two scenes in both domains do not match. So if the method does not color the point cloud well, it will also map an incorrect class prediction onto the 3D point cloud, even if the class is predicted correctly in the 2D images. Therefore, the points on the cyclist are likely to be labelled as *road* or *car* in 3D.

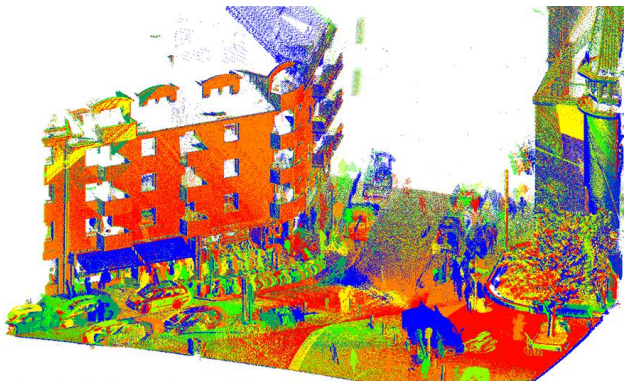


*Figure 4.4: Aligned point clouds in which each color indicates a different mapping campaign. Image source Schön et al. (2018)*

To tackle this problem, a method introduced by Peters and Brenner (2019) is used to detect such cases. A requirement for this method is that every scene is measured multiple times in different mapping campaigns, see Fig 4.4. As described by Brenner (2016), the point clouds from different dates can be aligned using a Hadoop computing cluster. It shows a scene colored by the capture date $r \in \{1,\ldots,R\}$, where $R$ is the number of all campaigns that are part of the dataset. Of course, moving objects will not appear in the same place at different times. Therefore, the alignment can give an indication of how static an object is, depending on how often it has been measured in the same scene. However, at this stage of processing, there is no information about which 3D points belong to the same object or class. The points from different epochs can be compared based on their spatial relationship to each other. In this case, the entire aligned point cloud is voxelized to group points from different mapping campaigns in space. Under the assumption that a voxel which was occupied in all campaigns refers to a static object and a voxel occupied only once belongs to a moving object, a feature can be generated to detect if a point is dynamic. Accordingly, the campaign count $\xi$ is defined as the number of mapping campaigns from which a point is assigned to a voxel. This value is assigned to each point that is present in the corresponding voxel. After normalization of $\xi$ to [0,1] by the maximum number of mapping campaigns $R$ a low value near zero corresponds to very dynamic points and a high value near one corresponds to static objects. Image 4.3c shows the campaign count $\xi$ per 3D point for the according scene 4.3. In this image, static points are colored bright red and dynamic points are gradually darker. On the one hand, this feature helps to easily remove the (dynamic) cyclist that has received a $\xi$ value near zero (dark color). On the other hand, the parked cars are preserved in the background because they were hit several times during different measurement campaigns.

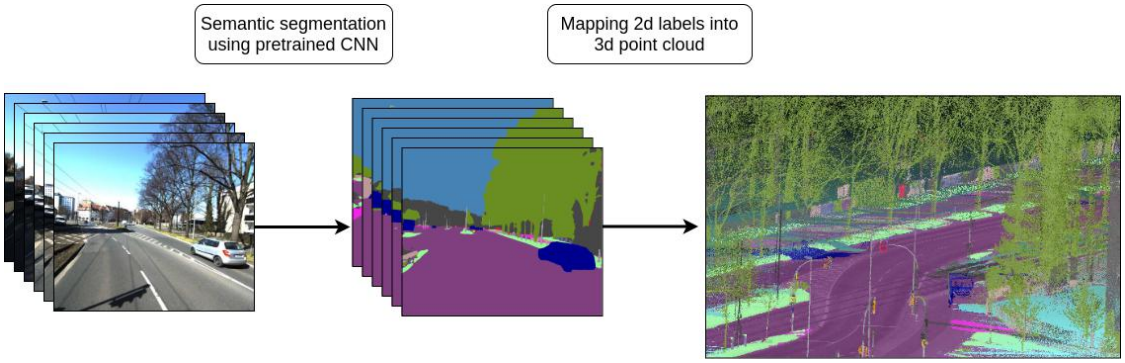



*Figure 4.5: Schematic example of the point cloud annotation process. The labels on the point cloud are assigned with majority vote.*

### 4.1.3 Naive Label Transfer and Label Policy-Based Noise

Naive label transfer describes the annotation of a point cloud from a set of multi-view images (Fig. 4.1 Box 4.I.) without using learning based methods to handle wrong assignments. In general, as shown in the example in Fig. 4.5, each image $I_k$ is semantically segmented by a pretrained DCNN $F(I_k) = \hat{y}_k$, where $\hat{y}_k$ is semantic segmented image. By projecting each 3D point $p_j$ into each semantically segmented image $\hat{y}_k$, each point can be assigned to a list of classified pixels $\hat{l}_j = \{\hat{y}_1(u_1,v_1),\ldots,\hat{y}_k(u_k,v_k)\}, k \in \{1,\ldots,K_j\}$. Because each 3D point $p_j$ can be projected into $K_j$ images, it is also possible that different labels are assigned to this point. To accumulate these labels into a fixed sized vector, a histogram $h_j$ is defined. The number of bins for each histogram are the classes that are predictable by the pretrained DCNN. When a class $c = \{1,\ldots,C\}$ is observed, the count inside the bin $h_{j,c}$ is increased by one. The resulting histogram may contain contradictory information; for example, if a vehicle passes through the scene when the images were taken, the histogram will contain non zero entries for *car* and *road* labels. To some degree, the ray tracer and majority voting on the histogram will mitigate these types of error by choosing the class with the greatest number of votes. However, this only applies if the errors belong to the minority classes. Otherwise, they will introduce label noise, which describes the effect that 3D points will be assigned a wrong class label. Another reason for such errors as those already mentioned is label noise due to conflicting label policies

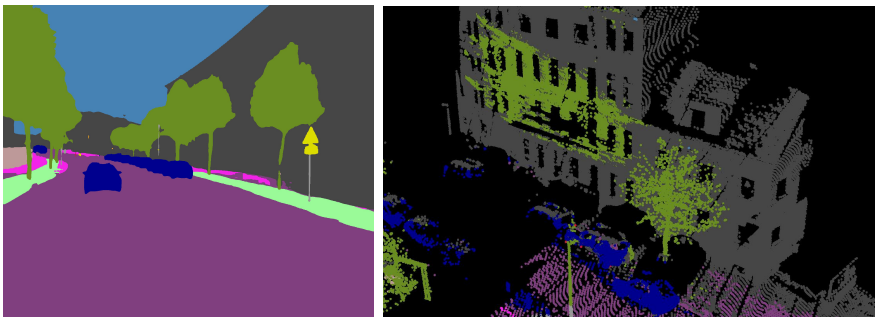


*Figure 4.6: An example of label noise due to Cityscapes labeling policy. The left image shows a semantically segmented MMS image. The right image shows a point cloud with classes assigned to the majority vote, where the facade was incorrectly assigned to vegetation. Image source (Peters and Brenner, 2019).*

The Cityscapes labeling policy states that areas in images that are visible behind tree canopies, such as building facades, are assigned to the class *vegetation*. When projecting 3D points into the images, points can appear in the image that are located behind the tree tops, assigning *vegetation*

labels to the points on the facade. The example in Figure 4.6 shows how this effect results in *vegetation* labels that are incorrectly assigned to the building behind them. However, since the car drives through the scene and also captures multi-view images from other perspectives, the histograms for these 3D points also collect class votes for *building*. Even if the majority decision leads to the label *vegetation*, this creates a certain pattern in the histograms that can be used to detect this type of error.

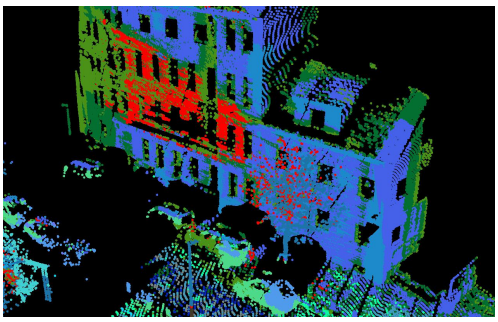


*Figure 4.7: Histograms clustered with k-means clustering. Classes are randomly colored. The red points on the building correspond to histograms containing vegetation and building labels. Image source (Peters and Brenner, 2019).*

The image in Fig. 4.7 shows the result of k-means clustering of histograms with $k = 32$ classes. Each histogram was assigned to the nearest centroid and colored according to the cluster number. The bright red points on the building show that it is possible to detect histograms with two or more classes. This confirms that the histogram $h$ can be used as a feature to find out whether the majority decision is likely to be correct in order to remove or correct this type of error.

#### Conclusion

In this section, a framework for transferring labels from images to point clouds was presented. After some origins of label noise were identified, new features were introduced to be used to correct or remove incorrectly assigned labels at a later stage. The framework has the potential to generate very large amounts of labelled point cloud data. This data can be used to train deep neural networks to learn semantic segmentation of the point cloud or to detect objects in 3D.

## 4.2 Label Noise Correction

The previous sections described the general procedure for labelling 3D points and possible sources for label noise. In addition, various methods were presented to detect or mitigate occlusion-related problems in label transfer. This section presents the methods for correcting label noise in 3D. All methods are learning based, i.e. the general procedure is to learn to correct or remove the wrong class labels. For this purpose, small reference sets of human annotated data in 2D and 3D are available.

### 4.2.1 Scanstrip-Based Noise Correction

In this scenario, 3D point clouds are represented as scanstrips, see Fig. 2.8a. One way to clean the label noise after the transfer is to learn how to correct the assigned labels in a supervised manner. In this case, a classifier $F(x) \rightarrow y$ must be trained to map from a scanstrip $x$ to the

correct label-map $y$. A process which is also called "correction" or "cleansing". In the following all methods will have the following features available for every 3D point:

- The histogram $h_j$ containing the accumulated labels after the transfer from the classified 2D images. Please note that ray tracing will always be used to mitigate occlusions in the transfer step.
- The campaign count $\xi_j$ which gives an estimate of how dynamic the point is.
- The reflectance $r_j$ which was measured by the laser scanner. This value will be globally normalized between zero and one.
- The measured distance $\delta_j$ by the laser scanner. This value will also be globally normalized between zero and one.
- The estimated normal vector $\vec{n}_j$.

### Point-Wise Correction

In its simplest form label noise correction can be achieved by flattening each scanstrip and training a simple machine learning model like Gradient-Boosted Decision Trees (GBDT) that treats every point independently. This serves as a baseline for label noise correction, because the GBDT will not take the context into account. It also can be trained well in a reasonable amount of time, and is also relatively easy to tune. The results of the training will be discussed in Chapter 7.4.1. The tree is trained with different combinations of the available features to assess the importance and influence of each feature in the training result. This knowledge can then be applied to the procedure in the next section. It also serves as a basis for the following methods to gain insight into how great the benefits of deep learning based methods are.

### CNN-Based Correction

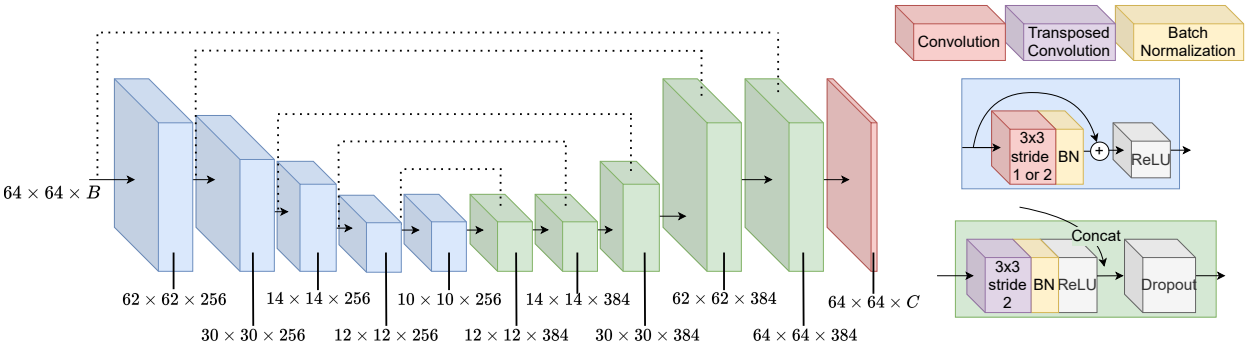



*Figure 4.8: Scanstrip Network (SNet). The network follows a U-Net structure (Ronneberger et al., 2015) with residual blocks. The kernel size is $3 \times 3$ in every convolutional layer. The dotted lines indicate the skip connections between the layers. The size of the feature channel is $B$ and the number of predicted classes is $C$.*

Unlike GBDT, the CNN-based correction will also contain the neighborings. The proposed network in this section is called Scanstrip Network (SNet) because it is trained directly on the scanstrips, similar to a traditional 2D CNN. SNet is based on the U-Net architecture by Ronneberger et al. (2015) with a few modifications. The general idea of the proposed network architecture is that SNet

should have lower model capacity in order to generalize well on only a few training examples[2], and that it should be very precise, since scanstrips have low resolution. The network receives as input $x_i$ a cropped scanstrip image with size $64 \times 64 \times B$, where $B$ is the number of features, see Figure 4.8. To achieve high resolution, the network performs only two downsampling steps. The corresponding output is a class map $\hat{y}_i$ with a resolution of $64 \times 64 \times C$ where $C$ is the number of possible classes. For higher input windows such as $128 \times 128$ or $256 \times 256$ no additional layer will be added but the stride will be increased for layer 3 or for layers 3 and 4. By default, U-Net increases the number of kernels incrementally, starting with 64 for the first layer, to extract low-level features at the beginning of the input. Since here the input already contains high-level features, the number of kernels in the encoder network is increased to 256 instead of 64 in U-Net. Also, each convolutional block in U-Net is exchanged with a residual connection, as shown in Figure 4.8. This means that a skip connection exists after each layer to deal with vanishing gradients and speed up training. Also, the max-pooling layer of U-Net is replaced by strided downsampling at the end of each residual block. The upsampling layers are reduced to contain only one transposed convolution, without any additional convolution after it, so that the network has a smaller model complexity.

The ground truth $y_i$ is given by a manually annotated scanstrip with the same size as the input $64 \times 64$. The scanstrip network is trained by minimizing the cross-entropy $H(y_i, \hat{y}_i)$ (Equation 2.16). Since not every pixel in the scanstrip is annotated, empty pixels will be removed from the loss function. The loss function is minimized using the Adam optimizer by Kingma and Ba (2015). As learning rate the the one by the authors of 0.001 is used. Additionally a dropout of 0.25 and random image flipping is used. Due to the scanstrip properties, the input window can be upside down when taken from the bottom half of the scanstrip, so it makes sense that the input is not only randomly flipped from left to right, but also upside down.

### 4.2.2 Semi-Supervised Scanstrip-Based Noise Correction

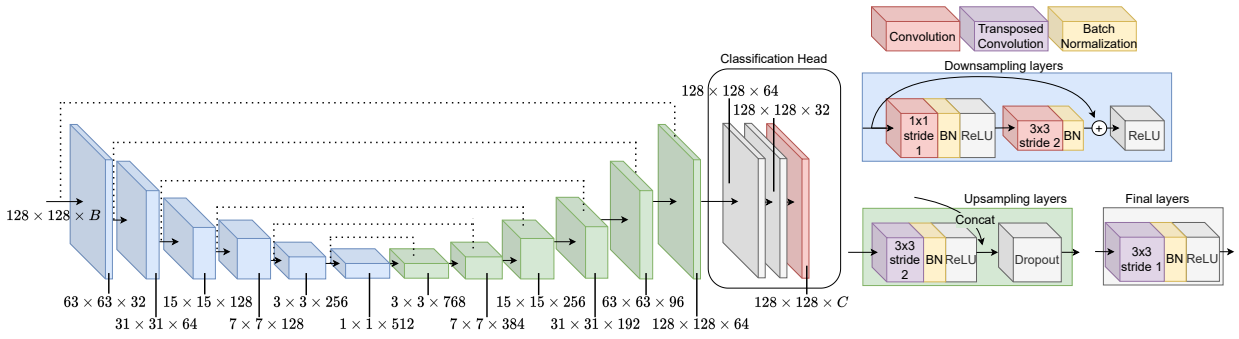



*Figure 4.9: An image showing the Scanstrip network $SNet_{SSL}$ in the first training phase of the semi-supervised learning approach. The input of the network are all features except the label histogram $h$. Here the label histogram serves as ground truth. The classification head is later used in the second training phase as seen in Fig. 4.10*

The naive label transfer method, described in Section 4.1.3, is theoretically capable of annotating an infinite amount of data with noisy labels. This property can be used to make the previous method semi-supervised to increase the prediction quality and obtain better results. The general training protocol is as follows: The training is done in two steps. First, the network in Figure 4.9 is trained using high volumes of noisy data that are automatically generated using naive label transfer. Then the network will be adapted and fine-tuned on the small human annotated reference set.

[2]SNet has only $\approx$ 4 million trainable parameters vs. for example a similar U-Net has $\approx$ 31 million

**The first step** consists of training the network on the entire dataset using the following loss function:

$$\mathcal{L}_{SSL,1} = H(\text{SNet}_{SSL,1}(\xi,r,\delta,\vec{n};\theta_1), \arg\max(h)) \tag{4.1}$$

Here $\text{SNet}_{SSL,1}(\xi,r,\delta,\vec{n})$ is the network shown in Figure 4.9 with trainable parameters $\theta_1$. The input for the network is a cropped scanstrip, the input of the network $\xi,r,\delta,\vec{n}$ describes the used features in the scanstrip. Accordingly, $\arg\max(h)$ describes a scanstrip with majority voting. The idea is that the network should learn features from all available data except from the histogram by minimizing the cross entropy $H(p,q)$ between the prediction and the noisy labels. The noisy labels are given by the $\arg\max(h)$ of the histogram $h$. Even if the labels are not correct, the task the network has to solve is identical to the actual one, namely to perform the semantic segmentation.

In the previous section, the network had access to the histogram as an input feature. Here, this is not possible because it would lead to a trivial solution, as the network would only need to create a peak where the histogram is at its highest value. This would prevent the network from learning any features. This is why $h$ is removed from the input, as shown in Equation 4.1, to force the network to learn feature extractors for campaign count $\xi$, reflection $r$, and normal vector $\vec{n}$. The overall network architecture for this approach is shown in Figure 4.9. It shows that the architecture of the network in Figure 4.8 has been changed to accommodate the new circumstances. As the network has no access to the histogram and only to low-level data, the network is deeper and has more downsampling and upsampling layers. In addition, the number of kernels per layer is gradually increased from 32 to 512, as is the case in most networks such as the U-net. Also, an additional layer has been added between the residual skip connections to make the network even deeper. Finally, a classification head was added containing three convolutional layers. This part will be used in the second step to obtain rich high resolution features for the final classification.

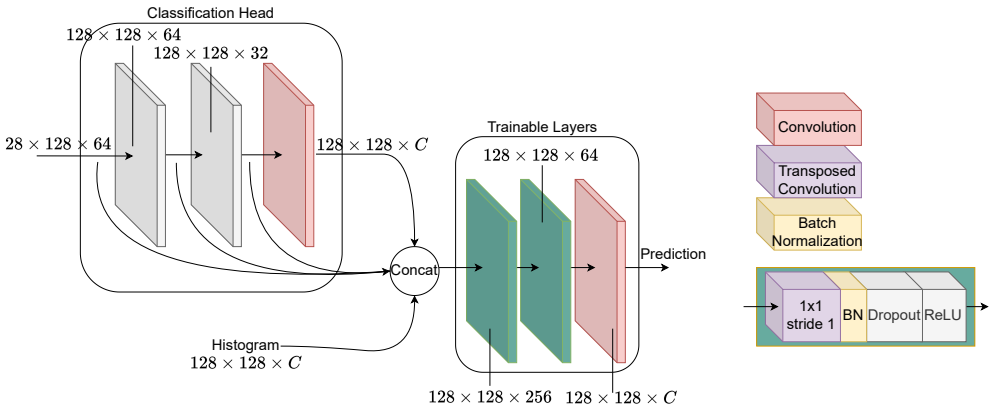



*Figure 4.10: In the second phase of the semi-supervised approach, all layers except the last three (dark green and light red) are frozen. The input of the last layers are the scanstrip containing only the histograms and the extracted features of the last four layers of the network from step one. All inputs are concatenated as shown in the figure and then passed to three successive convolutional layers, each of which has kernels of size* $1 \times 1$

**The second step** is to fine-tune the network using the human-annotated reference set. The difference to the procedure in the previous section is that the network $\text{SNet}_{SSL,1}$ has already learned feature extractors from the entire dataset, not just the training set. Fine-tuning is done by "freezing" all weights $\theta_1$ and adding three new layers along with the histogram $h$ as shown in Figure 4.10. The network in the second step $\text{SNet}_{SSL,2}$ will use the classification head along with the histogram to learn how to obtain correct predictions. Let $\Omega(F,l)$ be a function that returns the

feature map of layer $l \in \{1,\ldots,n\}$ of a network $F$ with $n$ layers. Let $\omega,\psi,\chi$ and $\phi$ be the feature maps of the last four layers of $\text{SNet}_{SSL,1}$ as follows:

$$\begin{aligned} \omega &= \Omega(\text{SNet}_{SSL,1}(\xi,r,\delta,\vec{n}),n) \\ \psi &= \Omega(\text{SNet}_{SSL,1}(\xi,r,\delta,\vec{n}),n-1) \\ \chi &= \Omega(\text{SNet}_{SSL,1}(\xi,r,\delta,\vec{n}),n-2) \\ \phi &= \Omega(\text{SNet}_{SSL,1}(\xi,r,\delta,\vec{n}),n-3) \end{aligned} \tag{4.2}$$

As the histogram $h$ may still contain valuable information, it is concatenated with the feature maps $\omega,\psi,\chi$ and $\phi$. The concatenation is indicated by the symbol $\oplus$. This is then fed to three successive convolutional layers $\text{SNet}_{SSL,2}$ with the third output representing the final label distribution, s. Fig. 4.10. The full equation for the new output $\hat{y}_{SSL}$ is as follows:

$$\hat{y}_{SSL} = \text{SNet}_{SSL,2}(h \oplus \omega \oplus \psi \oplus \chi \oplus \phi; \theta_2), \tag{4.3}$$

Where $\theta_2$ are the trainable weights of $\text{SNet}_{SSL,2}$. In this step, only the parameters $\theta_2$ are optimized, while $\theta_1$ remains untouched. The following equation is used for this training step:

$$\mathcal{L}_{SSL,2} = \mathcal{L}(y,\hat{y}_{SSL}) + \alpha\mathcal{L}_{\ell 2} \tag{4.4}$$

This loss computes the cross entropy between the prediction of $\text{SNet}_{SSL,2}$ and the human annotated ground truth label $y$. In addition, a regularization term $\mathcal{L}_{\ell 2}$ is added to prevent overfitting. The regularization term, sometimes referred to weight decay, is given by the following equation:

$$\mathcal{L}_{\ell 2} = \sum_{j=1}^{m} ||w_j||_2, \tag{4.5}$$

whereby $w_j$ is one of the trainable weights in $j \in 1,\ldots,m$ of $\text{SNet}_{SSL,2}$ and $\alpha$ is a scalar which is set to a fixed value of 0.00025. The updates are applied in both steps using Adam optimizer with the suggested standard hyperparameters.

### 4.2.3 Conclusion

In this section, a method for (naive) label transfer was presented. Furthermore, several causes for label noise were shown when pixels are naively assigned to a 3D point based on projection only. Possible reasons for a misclassified 3D point can be classification errors, an inaccurate labeling policy, regular or self-occlusions, and dynamic occlusions. To avoid simple occlusions, a filter based on ray tracing was introduced. Furthermore, features were shown that could potentially detect labeling policy problems and dynamic occlusions.

Furthermore, three learning based methods were shown that could be used to solve these problems. The first method is GBDT, which treats each point independently. It is mainly used as a baseline and to show which features are important. The next methods are fully convolutional neural networks that receive scanstrips as input. Both networks are roughly based on the U-net architecture, but are highly customized to the problem at hand. The first network specializes in using features in the form of class histograms in the scanstrip and has low model complexity. The method shown at the end is an extension of the first network. It is designed to use large amounts of automatically generated data to learn their representation and only small amounts of annotated data to learn the correct classes.

## 4.3 Multi-View Outlier Correction and Label Transfer

Looking at the previous approaches for classification of 3D data under label noise, they all deal with label noise only implicitly by correcting it after the predictions in the histograms have been aggregated. An early cause of label noise is incorrect classifications by the pretrained DCNN, see Fig. 4.1, box 2 to 3. The general problem is that a DCNN tends to overfit to the training data. For example, state-of-the-art networks can achieve up to 84.5% Intersection over Union (IoU) on the test set of the Cityscapes dataset (Sun et al., 2019). However, as Recht et al. (2018) and Recht et al. (2019) found, these benchmarks can result in highly specialized networks that even overfit to the test data. They measured a performance drop between 4% and 10% when testing pretrained networks on samples that were not present in the test set but were collected from the same domain or source. It is likely that this effect is even stronger when the DCNN is confronted with unknown data from a new domain. This case is sometimes referred to as domain gap, which describes the performance gap between the original test set and a new dataset. The straightforward solution of closing this gap by annotating a sufficient number of samples from the new domain is often too costly, especially when the annotation process involves pixel-by-pixel classification of thousands of images.

Peters et al. (2020) have shown that using multiple 2D images of the same object but from different views (multi-view) can be used to correct wrong predictions by learning to assign a consistent class to all observations. To do this, they used the procedure already presented in Section 4.1, to map a 3D point to images and find a list of corresponding pixel predictions in images that relate to that point in 3D. Their network architecture uses this list, along with other 2D and 3D features, to learn how to assign a consistent class to all these 2D image pixels. The problem they faced was mainly how to deal with this type of data structure, since the input is (1) unordered and of arbitrary length, and (2) uses 2D and 3D features simultaneously. After training their network, they were able to use the predictions of it as pseudo-labels to fine-tune the DCNN and close the domain gap. Although they were able to show that the fine-tuned DCNN performed better, their method mainly ignored (dynamic) occlusions and calibration errors because they assumed that all 2D pixels actually belong to the same object. In the following section, the Multi-View Network is proposed that can overcome this problem by accessing the same input data but making independent predictions for each pixel, thus accounting for occlusions and calibration errors.

### 4.3.1 Multi-View Network

Multi-View Networks (MVNets) are able to learn how to correct semantically segmented 2D images by having access to multiple views. The use of multiple views has two major advantages: First **multi-view analysis increases the amount of training data and helps model generalization**. Training a network using only very few ground truth images, may not be sufficient for the classifier to generalize well. However, by linking pixels from the reference set to pixels from other images within multiple views of the same point, the amount of data can be greatly increased, which can reduce model variance. Thus, whenever the Multi-View Network makes a prediction for a single 2D image pixel in the reference set, it simultaneously has access to all views of the same patch from different angles, heights and distances. This, together with the network design, creates a synergy because all the trainable parameters are shared in the Multi-View Network so that the network is also trained on the unannotated multi-view image data and eventually generalises better. Secondly **the network can learn to propagate predictions through space**. A classic DCNN like Deeplab or HRNet only has access to one image at a time. The Multi-View Network, on the other hand, can learn to transfer correct predictions from one image to another by using **self-attention**. As shown by Peters et al. (2020) a simple majority decision over multiple views can already impact the IoU of certain types of objects. With self-attention, the MVNet is able to

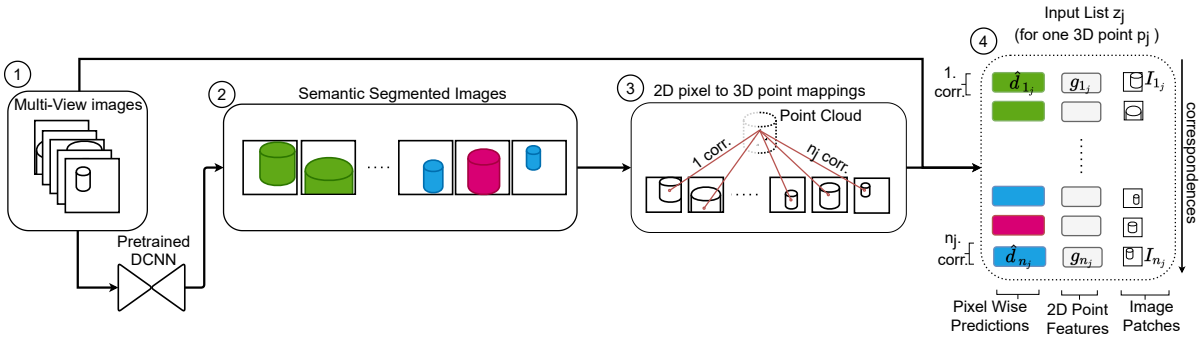



*Figure 4.11: Procedure for creating an input list for one 3D point. The input data are multi-view images (Box 1). Different colors in Box 2 indicating different classes per pixel. In Box 3 one 3D point $p_j$ is mapped to all images creating $n_j$ correspondences (corr). The resulting list $z_j$ for the 3D point $p_j$ with length $n_j$ is the input of MVNet. The length of the list corresponds to the number of multi-view images. Each entry contains the pixel wise prediction $\hat{d}_{n,j}$ for each of the n images, as well as 2D image features and RGB image patches cropped around each corresponding image pixel.*

relate all predictions for a point and can make independent predictions for each image, resulting in a higher IoU.

#### 4.3.1.1 Problem Definition and Data Structure

For one 3D point $p_j$ in $j \in \{1,\ldots,J\}$ one can assign a list of predicted class distributions $\hat{d}_j$ from $n_j$ corresponding semantically segmented multi-view images. The list $\hat{d}_j$ is aggregated by projecting a 3D point $p_j$ into $n_j$ images and combining the predicted class distributions at all back-projected pixel locations. Therefore $\hat{d}_j$ has a size of $n_j \times C$ where $C$ is the number of predictable classes. The general problem statement is: Under the assumption that $\hat{d}_j$ contains label noise, what is the actual list of classes $y_j$? Where $y_j$ has length $n_j$. Because all entries in $\hat{d}_j$ belong to the point $p_j$, probably some entries will correspond to the same object. However, some of the predictions may be wrong due to the domain gap or mismatched due to occlusions or calibration errors. For this a classifier will be introduced that is able to learn to map from $\hat{d}_j$ along with other features to the actual classes $y_j$.

Figure 4.11 shows how an input sample for the classifier is aggregated. Basically, it is a more detailed version of Figure 4.1, boxes 1, 2 and 3. It shows that each 3D point $p_j$ is possibly mapped to $n_j$ pixels $I_k(u_k,v_k)$, where $k \in \{1,\ldots,n_j\}$ and $u_k$,$v_k$ are the pixel coordinates. For each of these pixels, a pretrained DCNN will predict a class distribution resulting in list $\hat{d}_j$. As shown in Figure 4.11 (Box 4.), the resulting list $z_j$ contains, per entry, a predicted class distribution, some other features $g_j$ with size $n_j \times G$ corresponding to the pixel coordinate, and an RGB image patch of size $s \times s$ containing direct adjacent neighbours centrally cropped around each looked up image pixel $I_k(u_k,v_k)$.

#### Network Architecture

While most classification problems require a fixed input size and data which are ordered in some way, the generated list $z_j$ is of variable length and in no particular order. Even worse, the data contains no obvious spatial relationship that can be exploited with convolutional layers. The naive solution to this problem was shown in 4.2.1. Here, the predictions were accumulated in a histogram of fixed size. Another solution would be to reduce the input list $z_j$ to a fixed size by passing it to a reduction function. If $z_j$ has a size of $n_j \times Z$, where $n_j$ is the list length and $Z$ is the feature length,

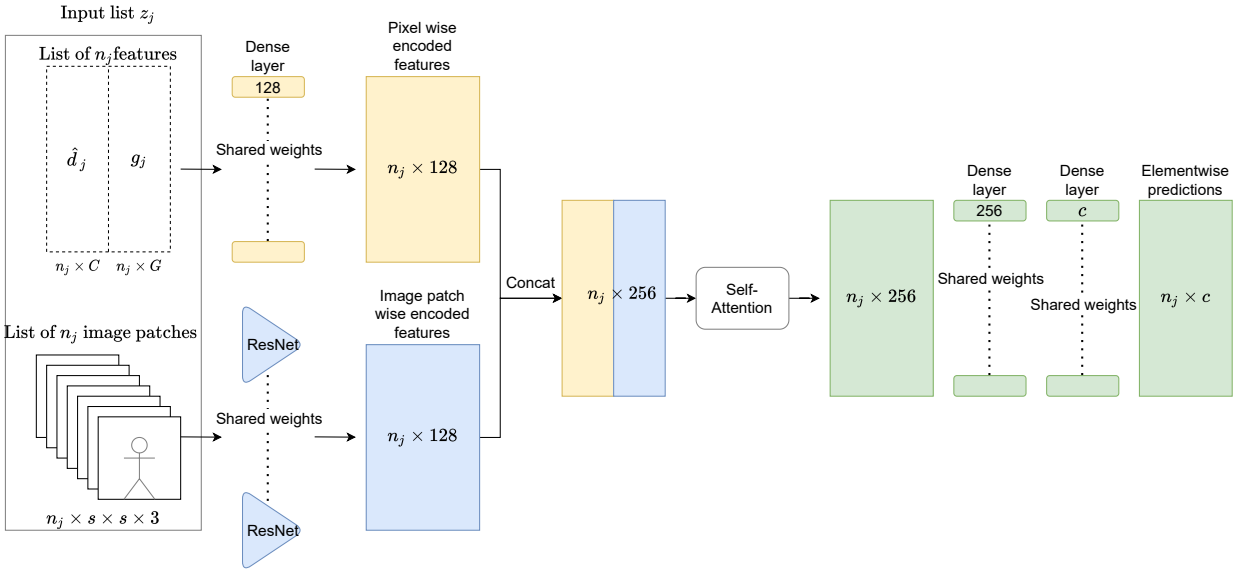



*Figure 4.12: The proposed network architecture for learning to predict a corrected list of labels with $C$ classes for a list of multi-view observations $z_j$. The weights of the dense layers (yellow and green blocks) are shared. However, they are applied to each list entry separately. The same is true for the ResNet networks (Blue triangles), they are applied to each image patch individually, but all their weights are shared. The joint output of all ResNet networks is an $n_j \times 1 \times 1 \times 128$ tensor with $1 \times 1 \times 128$ per image, which is then reshaped to $n_j \times 128$. All dense layers in the network use batch normalization and ReLU as activation.*

then it would be reduced to a fixed size vector $1 \times Z$. A function for this purpose could be, in the case of class distributions, the *sum* over all distributions. This would create a fixed-size vector favoring the class with the largest peaks in all predictions, which is very similar to a majority vote in the aggregated histograms in Section 4.2.1. The network presented by Peters et al. (2020) solved the input problem similarly to PointNet (Qi et al., 2017b). The network first embedded each list entry of $z_j$ in a high-dimensional latent space and then used a symmetric function along the first axis to reduce the encoded input to a fixed-size feature vector. This could then be fed into multiple dense layers to predict one final class distribution for all related multi-view image pixels. However, as described earlier, this would mean that the input list could not contain any mismatched pixels, because the predicted class distribution would also be assigned to those as well. Peters et al. (2020) avoided this problem partially by not correcting classes that belong to moving objects, which are very likely to include dynamic occlusions.

Another (naive) solution to correct the list $\hat{d}_j$ would be to ignore the fact that all entries are related and make an independent prediction for each entry. Because a network would only make predictions based on individual viewpoints, the corresponding network will be called Single-View Network (SVNet) in the following. To take advantage of the information that all predictions are related, the solution presented here is based on a slightly different approach. Instead of making independent predictions or reducing the input list to a fixed-size feature vector, here all features are related using self-attention, see Fig. 4.12. This allows the following dense layer to consider all multi-view observations together but still provide an individual prediction for each observation. The multi-view observations of the input list $z_j$ are combined as shown in Fig. 4.12. The pixel-wise predictions $\hat{d}_j$ and 2D point features $g_j$ are fed into $n_j$ dense layers with 128 filters (yellow blocks) that share their weights and process each entry individually. The corresponding image patches are fed to $n_j$ ResNets (blue triangles), which also share their weights and process each patch individually. The network structure for the ResNets is shown in Figure 4.13. Depending on the size

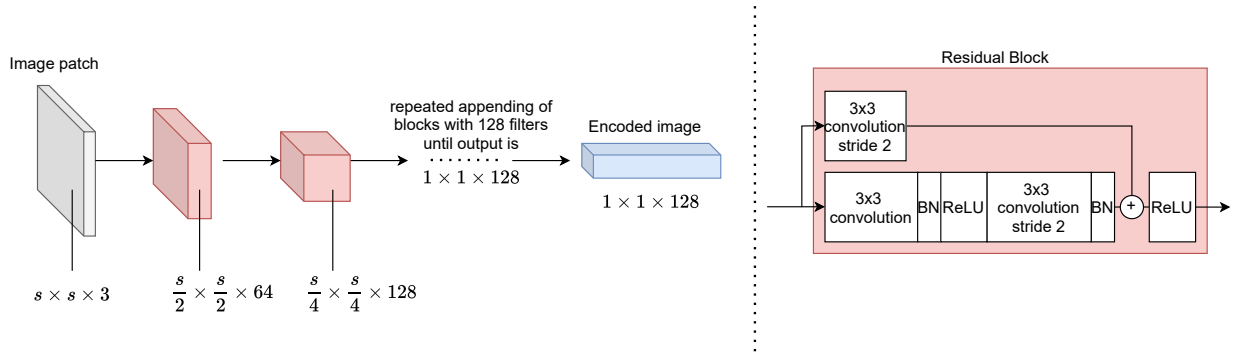



*Figure 4.13: The ResNet block from Figure 4.12. This subnet repeatedly uses residual blocks with a striding of two until the input image patch is encoded in a feature vector of size* $1 \times 1 \times 128$*. The residual blocks are described in the right part of the image, where BN stands for batch normalization.*

of the image patch, each ResNet repeatedly uses residual blocks until the final feature output size is $1 \times 1 \times 128$ for an image. Using all ResNets, this creates a feature vector of $n_j \times 1 \times 1 \times 128$, which can be reshaped to $n_j \times 128$ (blue rectangle). Both blocks now contain separately encoded feature vectors, one for each view. These vectors are concatenated into a tensor of size $n_j \times 256$ and fed to the self-attention module. The self-attention module has a key and value size of $k = 64$ and $v = 64$, see Equation 2.39. The output of the self-attention module now contains features that are related to each other, meaning that a single observation can influence the outcome of another observation. This list is then passed to two successive dense layers to output the final class distribution.

The training of the Multi-View Network MVNet($z_j$) can be done end-to-end using cross entropy:

$$\mathcal{L}_j = H(\text{MVNet}(z_j), y_j), \tag{4.6}$$

Where for each input list $z_j$ a list of ground truth labels $y_j$ exits. The labels can be created by annotating real images and looking up the annotations at the same pixel coordinates pointed to by each entry in the input list. The training hyperparameters are introduced in the experimental chapter.

#### 4.3.1.2 Fine-tune a DCNN Using Pseudo-Labels by MVNet

The problem with MVNet is that it can do only predictions for points associated with 3D points. To fix this problem the predictions generated by MVNet can be used as pseudo-labels for fine-tuning a pretrained DCNN on the MMS images. Later experiments will show that this method gives better results than fine-tuning the DCNN directly on the labels on which the MVNet was trained. The procedure is carried out in the following steps:

- Step 1 : The pretrained DCNN makes an initial prediction for all images.
- Step 2: The images and semantically segmented images are matched with a 3D point cloud to create the multi-view data structure, as explained in Figure 4.11.
- Step 3: The MVNet is trained as described in the previous section using the input features $z_j$ and corresponding labels $y_j$.
- Step 4: The MVNet corrects all initial predictions associated with a 3D point. Note that due to sparsity of the point cloud, only image pixels associated with a 3D point are corrected.

– Step 5: The pretrained DCNN from Step 1 is fined-tuned on the corrected pseudo labels by MVNet for static classes. For dynamic classes the original predictions will be used as pseudo labels, because 2D to 3D point mappings suffer from dynamic occlusions.

This procedure is very similar to the one described by Peters et al. (2020). They created the pseudo-labels by replacing all initial predictions that do not belong to a dynamic class with the new corrected class prediction. The strategy here is adapted because MVNet$(z_j)$ estimates a list of class distributions and not not only a single class for the whole input list. The new pseudo label will be therefore crafted in the following way:

$$\hat{y}_j(k) = \begin{cases} \hat{d}_j(k) & \text{if } \arg\max(\hat{d}_j)(k) \in \text{dynamic} \\ \text{MVNet}(z_j(k)) & \text{otherwise.} \end{cases} \tag{4.7}$$

Note that the notation is adopted from Section 4.3.1.1. Where $\hat{y}_j(k)$ describes the $k$-th entry with $k \in \{1, \ldots, n_j\}$ of the $j$-th input list. Furthermore $\hat{d}_j(k)$ denotes the predicted class distribution of the pretrained DCNN and MVNet$(z_j(k))$ the predicted class distribution by MVNet for the same entry as $\hat{y}_j(k)$.

The procedure will be used and tested in Section 7.5.7 and 8.1.4.

### 4.3.2 Label Transfer Network

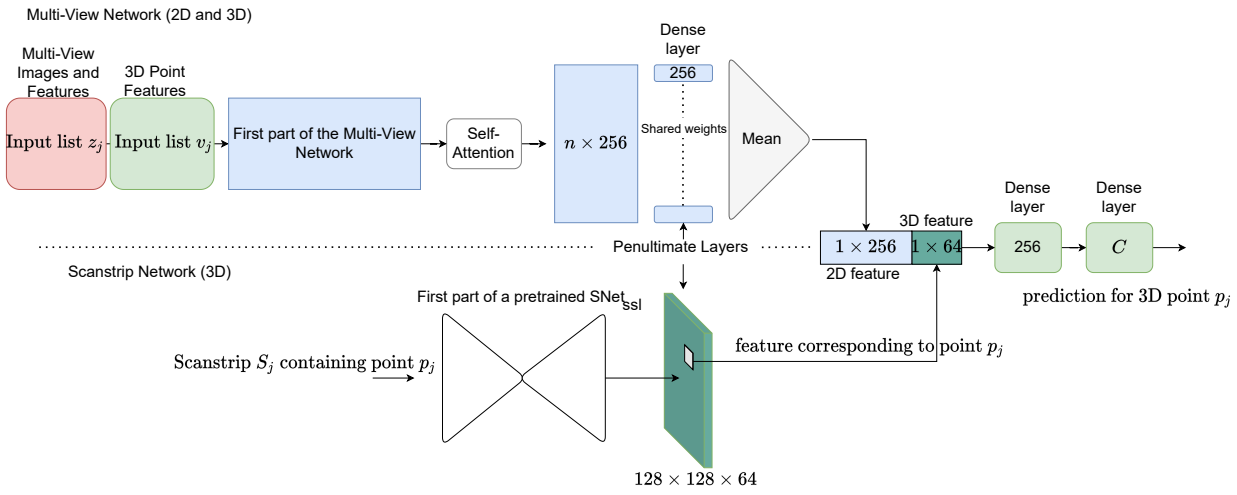



*Figure 4.14: A schematic overview of the Label Transfer Network (LTNet). The upper part of the network corresponds to the MVNet shown in Figure 4.12 up to the penultimate layer. The difference here is that MVNet receives also a list of 3D point features $v_j$. The extracted features of the MVNet are reduced to a fixed-size vector using average pooling. The lower part of the network is the same as in the semi-supervised scanstrip network (SNet$_{SSL}$), Figure 4.10. SNet$_{SSL}$ is pretrained and the weights are frozen during the training of LTNet. The feature of the penultimate layer of the SNet$_{SSL}$ corresponding to the 3D point $p_j$ is appended to the reduced multi-view feature vector. Both features are then used in conjunction to predict the final class for $p_j$*

The Label Transfer Network (LTNet) is a combination of all previous networks mentioned in this chapter. It is used to solve the problem defined in Figure 4.1. Box 4 I. It learns how to assign a class to a 3D point given a list of observations from multiple views and a local 3D point cloud. LTNet learns how to transfer labels given the multi-view observation using MVNet and the local 3D scanstrip neighborhood using SNet or SNet$_{SSL}$ as feature extractor, see Fig. 4.14. The network requires only a small set of 3D reference labels, which are the same as for SNet and SNet$_{SSL}$.

Although LTNet also uses 2D image data as input, no reference is used here. The schematic overview in Figure 4.14 shows that both networks are combined by concatenating the features of the penultimate layers and passing them to two dense layers in order to classify a single 3D point. The network can therefore learn how to explicitly account for each individual cause of label noise errors early on, such as occlusions, calibration errors, or label policy issues.

The network LTNet$(z_j,v_j,\Omega(\text{SNet}_{SSL},n-1)_j)$ has three distinct inputs. The first two inputs $z_j$ and $v_j$ are two lists of length $n_j$ which are passed to the MVNet. Whereby $z_j$ was already described in Section 4.3.1.1. The input list $v_j$ contains 3D features corresponding to the 3D point $p_j$. The 3D features used in $v_j$ are the campaign count $\xi_j$, the reflectance $r_j$, and the normal vector $\vec{n}_j$ of the 3D point $p_j$. Note that since there is only one 3D point that relates to $n_j$ images, the 3D features are simply copied $n_j$ times so that they are available for each entry in $v_j$. The third input feature for LTNet is $\Omega(\text{SNet}_{SSL},n-1)_j$. It has a size of $1\times 64$ and contains the feature vector corresponding to the point $p_j$ extracted from the feature map of the penultimate layer of the pretrained $\text{SNet}_{SSL}$. In order to fuse both domains the output of size $n_j \times 256$ of the Multi-View Network is reduced to a fixed size feature vector of $1 \times 256$ using average pooling. As can be seen in Figure 4.14 both feature vectors are concatenated to a feature vector of size $1 \times 320$ which is then passed to two dense layers to produce the final class probability distribution for point $p_j$ with $C$ classes.

During training, only the weights of the Multi-View Network and the dense layers in the end are updated, which has the advantage that the network is less prone to overfitting and training and inference are much faster because the feature vector $\Omega(\text{SNet}_{SSL},n-1)_j$ can be easily precomputed and stored. The entire network is trained in the same way as the SNets, i.e., the same labels are used.

### 4.3.3 Conclusion

In this section, a new network structure was presented that is capable of processing unordered multi-view observations of arbitrary length that contain both 2D and 3D features. The goal of the network is to correct incorrect predictions in 2D multi-view images made by a DCNN that suffers from a domain gap. The network does this by learning to encode each observation individually with two subnetworks and then relates them using self-attention.

In addition, the LTNet was introduced, which is a direct extension of the $\text{SNet}_{SSL}$ presented in Section 4.2.2. Instead of learning how to correct the data after the transfer, it has access to the complete label transfer pipeline by fusing the multi-view observations from MVNet with the 3D observations of SNet. It can therefore learn to ignore predictions in 2D that do not match the local 3D object, such as in dynamical occlusions.

# 5 Self-Supervised Point Cloud Rendering and Completion

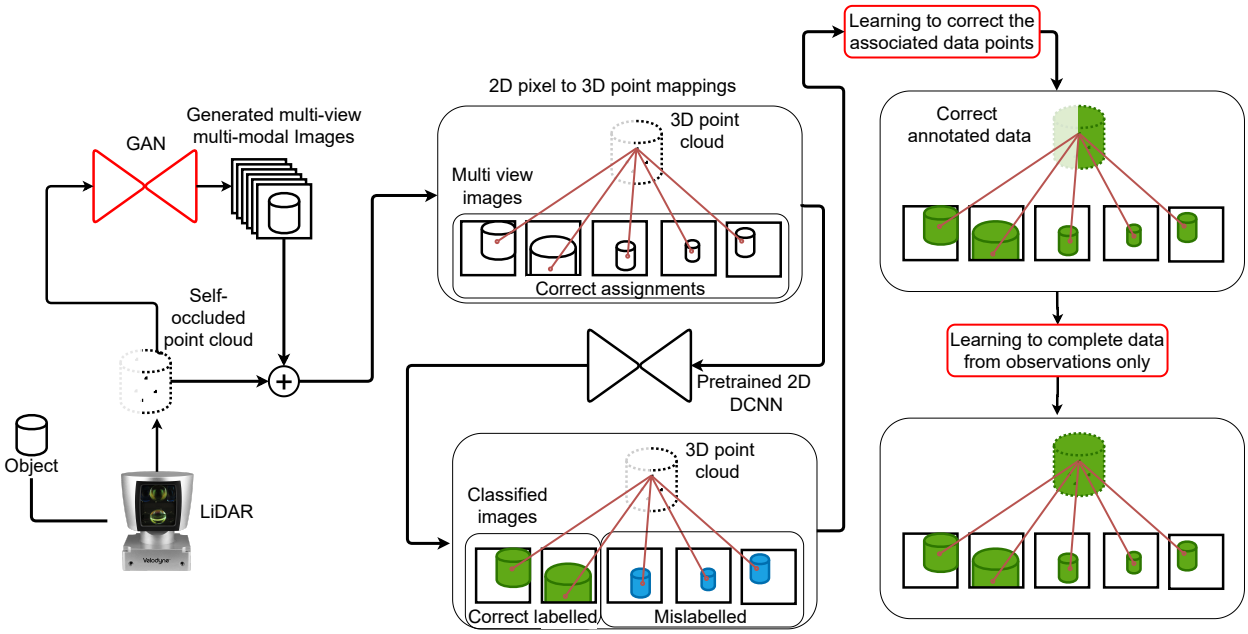



*Figure 5.1: This diagram shows an extension to Figure 4.1. In this Figure the MMS is mainly replaced by a GAN which estimates multi-view and multi-modal images based on the measured point clouds by the LiDAR. Additionally a step for resolving self-occlusions is added in the end.*

Naive label transfer suffers from label noise in 3D due to various types of occlusions. In this chapter calibration errors, dynamic occlusions and self-occlusions are treated with two self-supervised methods in order to improve naive label transfer. First, in Section 5.1 a GAN is presented that, once trained, is capable of estimating multi-view multi-modal photorealistic images only from point cloud data replacing the camera and GNSS/IMU in the MMS. As these images can be semantically segmented by a pretrained DCNN, the GAN can serve as an "interface" for transferring the 2D labels into the 3D domain, s. Fig 5.1. Besides the advantage of replacing a fully calibrated MMS with a GAN-based estimator, this GAN can also reduce label noise due to calibration errors and dynamic occlusions. Remember that dynamic occlusions only occur because camera and laser beams do not coincide. If an image is generated purely based on point cloud data, the images are very accurate in terms of matching the real point cloud, reducing occlusion and calibration problems.

Secondly, another GAN is presented in Section 5.2 that is able to learn the completion of object instances only from incomplete observations. As one reason for errors in label transfer is self-occlusion, this GAN can help to reduce label noise by providing an estimate of which unknown areas may be occupied by objects. When a label is transferred using ray tracing, this estimate can help to reduce "label bleeding" as it can block rays behind self-occluded objects. Both processes are fully self-supervised, which means that no human annotated reference data will be used.

## 5.1 Photo-Realistic Point Cloud Rendering

In this section a CGAN will be introduced that is able to create photo-realistic point cloud renderings. Point cloud rendering or visualization can be done by projecting 3D point clouds into virtual images. But this would not produce realistic images in such a way that they have similar characteristics to real images taken with a real digital camera. The goal of this method is to create images that are interpretable by a DCNN trained exclusively on a publicly available real image dataset. The degree of realism is therefore given by how the images can be semantically segmented by a pretrained state-of-the-art semantic segmentation network. A generative adversarial network is perfect for this purpose, because it bypasses traditional rendering methods and generates images that are realistic according to a discriminator, a DCNN that compares the feature composition in the "real" and generated images.

Rendering photorealistic images from point clouds boils down to a CGAN learning to predict camera images from projected point clouds. This requires a sufficiently large dataset. The trained CGAN can then be used to generate images for point clouds that do not have access to a fully calibrated MMS. As point clouds do not contain color information, the estimated images are colored mainly according to the object shape and the intensity of the reflected laser beam. However, a real image may look completely different if it was taken at different times of the day because the illumination changes. Different weather conditions and seasons also change the appearance of the images. If the generator cannot detect features for lighting or seasons in the point cloud, the generated images have a higher degree of freedom because any possible outcome for the same input is a valid representation for the discriminator. In order to produce consistent point cloud renderings, a method is presented to make the look of the generated images controllable. This is done by parameterizing the acquisition date of each image. The overall appearance of a rendering can then be modified and controlled by adjusting the capture date. Mapping from one input to many outputs is referred to as multimodality. This not only results in a uniform appearance, but also allows control over the appearance of the generated images.

### 5.1.1 Network Architecture

A common conditional GAN (CGAN) is trained on tuples $(x_i,y_i)$, where the generator $G(x_i) \rightarrow y_i$ maps a sample $x_i$ from the source distribution $X$ to a sample $y_i$ from the target distribution $Y$. In this particular case, the CGAN should map from a point cloud $x_i$ to a corresponding image $y_i$ which was recorded at a specific time $s_i$. As shown in Chapter 2.1.2.1, there are different ways to represent a point cloud, e.g. volumetric, as coordinate list or projected. The problem with the first two representations is that they do not encode the pose of the target image. This problem is solved by projecting the point cloud into the camera plane that corresponds to the pose of the target image, resulting in a 2.5D representation of the point cloud. The projected point cloud image contains two channels. The first channel stores the distance between each 3D point and the camera center and the second channel the reflectance. If more than one 3D point is mapped to the same 2D pixel the one with the shortest distance is kept because the others are occluded. There is no ray tracing implemented to take occlusions into account.

The generator network takes as input the projected point cloud image $x_i$ and a date $s_i$ which encodes the time when the target image $y_i$ was taken. The architecture of the generator as seen in Fig. 5.2 is similar to that of Johnson et al. (2016). Instead of a typical U-Net structure like the pix2pix presented in Isola et al. (2017), this network uses residual connections, which, according to Johnson et al. (2016), have the property of learning an identity function; This is an advantageous property for networks that have to transform between images, because they often share a similar structure. Empirically this has been confirmed by Wang et al. (2018a). They based their work

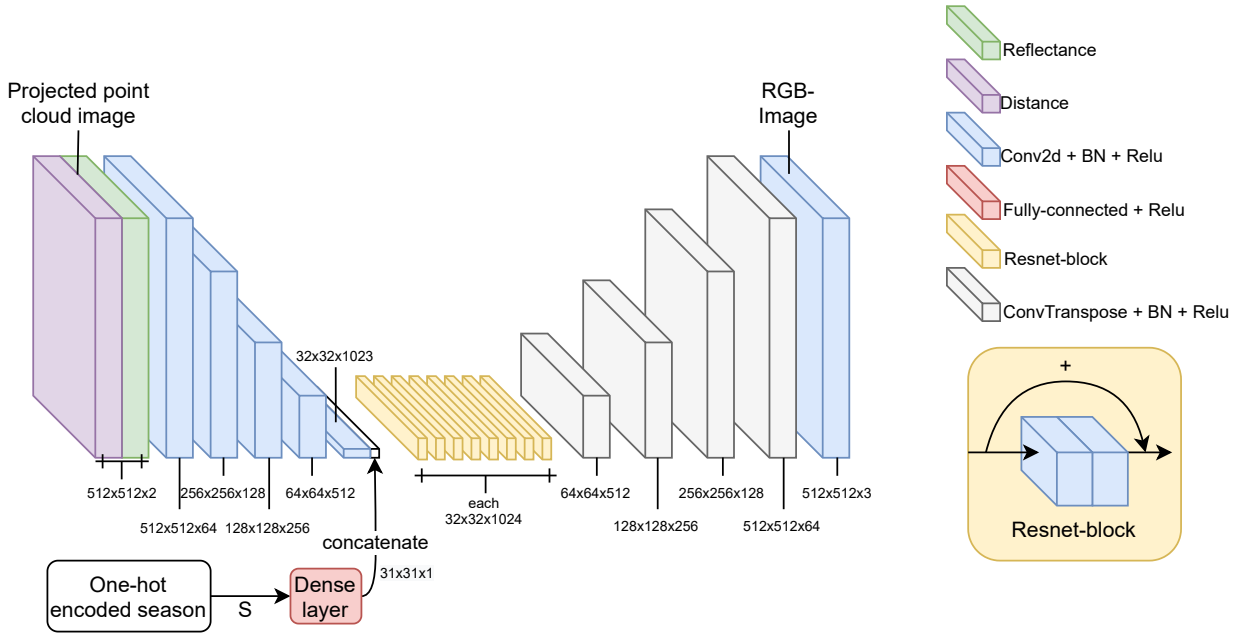



*Figure 5.2: The generator network receives a projected point cloud image with two channels and creates a photorealistic RGB-image. A one-hot encoded season vector of size $S$ is passed to a fully connected layer (red), whose output is reshaped to $1 \times 1 \times 1024$ and concatenated to the bottleneck (blue).*

on this architecture to outperform pix2pix in resolution and image quality. However, the network in Fig. 5.2 is adapted to the respective problem: The generator takes an input image of size $512 \times 512 \times 2$ and downsamples it four times with a strided convolution of two. The bottleneck (the smallest feature map) reaches a size of $32 \times 32 \times 1023$. Please note that the image in Figure 5.2 shows a bottleneck with 1024 features, the missing last channel will be described in the following.

Adding the date $s_i$ as image with constant value to the input as third channel is ignored by the generator. As shown by Peters and Brenner (2020), the date information $s_i$ can be added with a fully connected layer by concatenating it as feature to the bottleneck, s. Fig 5.2. The fully connected layer (shown in red) has as input a one-hot coded vector of length $S$, where $S$ indicates the number of different capture days in the entire dataset. The output of the fully connected layer has a size of 1024, which is resized to $32 \times 32 \times 1$ and concatenated to the bottleneck so that the combined feature map has a size of $32 \times 32 \times 1024$. Unlike Johnson et al. (2016), this network has four convolutions with stride two, followed by eight residual blocks with a feature size of 1024 each and finally four transposed convolutions. The original architecture is smaller and uses only three convolutions with stride two followed by five residual blocks (with a features size of 128) and three transposed convolutions.

The discriminator shown in 5.3 is based on the multi-scale discriminator introduced by Wang et al. (2018a). Each discriminator $\{D1,D2\}$ works on a different image scale, $D1$ on the original image scale $512 \times 512 \times 5$ and $D2$ on $256 \times 256 \times 5$ which is created by bilinear interpolation. The complete input to the discriminators is created by concatenating an RGB image (the real or fake one) and the corresponding projected point cloud. This is done to make the GAN conditional so that the generator is forced to react appropriately to the input image. Instead of a single scalar, the result of the discriminators is a $64 \times 64 \times 1$ and $32 \times 32 \times 1$ map, respectively, a strategy introduced by Isola et al. (2017) and referred to as patchGAN. The advantage of using a patchGAN is that the individual predictions of the discriminator can only be propagated back the corresponding support window (patches) of the input within the receptive field of the discriminator. Consequently the

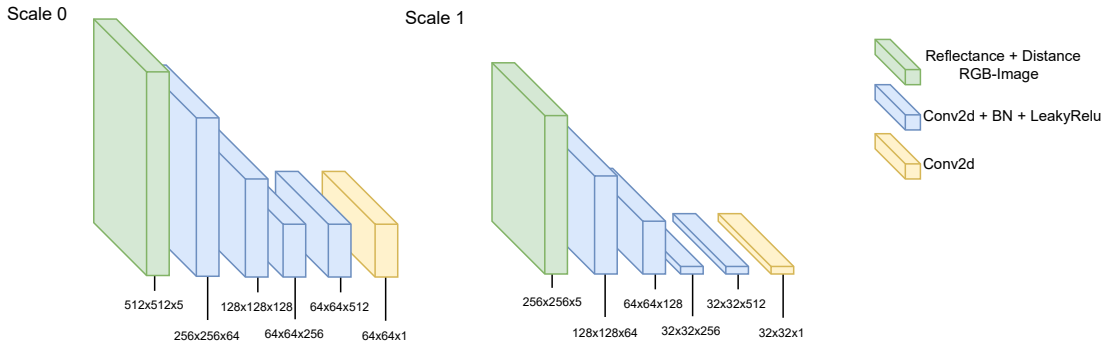



*Figure 5.3: The multi-scale discriminator networks.*

discriminator requires fewer parameters and runs faster. Empirically, this can also lead to a higher quality in the synthesized images, because each generated image field is independently evaluated by the discriminator.

It is noteworthy that the pix2pixhd GAN by Wang et al. (2018a) is similar to this GAN. However, pix2pixhd allows human interaction in order to manipulate generated objects. This is done by integrating ground truth data about instances and labels maps to know where an object is placed in the image. This gives the network the ability to encode object properties and to change the appearance of certain object instances. However, this is only possible with a fully annotated dataset that has access to pixel and instance maps such as Cityscapes (Cordts et al., 2016). Since the GAN presented in this section is self-supervised, the parts that encode the object appearance are removed from the architecture and instead the fully connected layer is added to the bottleneck to encode the acquisition date of each image. Additionally, due to hardware limitations, the generator network is changed as shown in Figure 5.2, which limits the image resolution to $512 \times 512$ instead of the original $2048 \times 1024$ of pix2pixhd. However, the image resolution is not so important in this case, because the image size can be increased later by stitching different generated image patches. As the CGAN can generate images for any camera position, this allows the various image patches to be stitched together to form a high-resolution image, shown in Section 5.1.3

### 5.1.2 Loss Function

Instead of the classical GAN loss function as shown in Equation 2.40, this CGAN is trained using the LSGAN loss introduced by Mao et al. (2017):

$$\begin{aligned} \mathcal{L}_{GAN}(G) &= \frac{1}{2}\mathbb{E}[(D(G(x,s),x) - 1)^2] \\ \mathcal{L}_{GAN}(D) &= \frac{1}{2}\mathbb{E}[(D(y,x) - 1)^2] + \frac{1}{2}\mathbb{E}[(D(G(x,s),x))^2], \end{aligned} \tag{5.1}$$

where $G(\cdot)$ and $D(\cdot)$ denote the generator and the discriminator networks and $x$ denotes the projected point cloud image, $s$ denotes the acquisition date and $y$ denotes the target RGB-image. The discriminator is trained by minimizing $\mathcal{L}_{GAN}(D)$. As the CGAN uses more than one discriminator, the actual loss will be defined for all discriminators $D_k$ with $k \in \{1,\dots,P\}$ as follows:

$$\mathcal{L}_{GAN}(D_1,\dots,D_P) = \frac{1}{P}\sum_{k=1}^{P}[\frac{1}{2}\mathbb{E}[(D_k(y,x) - 1)^2] + \frac{1}{2}\mathbb{E}[(D_k(G(x,s),x))^2]] \tag{5.2}$$

For the generator it is very common to be guided by additional loss terms like for example the L1 or L2 loss between prediction and and real image so that the generator prefers predictions which are similar to the actual image (Isola et al., 2017). Like pix2pixhd the generator uses a combination of the following loss functions:

$$\mathcal{L}(G) = \mathcal{L}_{GAN}(G) + \mathcal{L}_{vgg}(G) + \mathcal{L}_{feat}(G), \tag{5.3}$$

where $\mathcal{L}_{GAN}(G)$ is the LSGAN function defined in Equation 5.1. The additional functions $\mathcal{L}_{vgg}(G)$ and $\mathcal{L}_{feat}(G)$ help to increase the quality of the generated images. As mentioned before, the generator should produce images in which the features in the "real" and generated "fake" images are very similar. In the $\mathcal{L}_{vgg}$ loss the distance between the generated "fake" image and the "real" one is minimized in latent space. This is done by using a pretrained vgg classification network by Simonyan and Zisserman (2015) and comparing the detected features between both images. In the following $VGG^{(i)}(\cdot)$ is a function that gives the output vector of the output of i-th layer with a size of $M_i$ of the pretrained VGG network.

$$\mathcal{L}_{vgg} = \alpha \sum_{i=1}^{N} \frac{1}{M_i} ||VGG^{(i)}(y) - VGG^{(i)}(G(x,s))||_1 \tag{5.4}$$

The value $\alpha$ is set to 10 as it has been done by Wang et al. (2018a). The feature matching loss $\mathcal{L}_{feat}(G)$ is used to minimize the distance between the real and fake images in latent space. This is done extracting the features for the real and fake images from all discriminators and calculating the L1-norm between them.

$$\mathcal{L}_{feat} = \frac{1}{P} \sum_{k=1}^{P} \sum_{i=1}^{T} \frac{1}{N_i} ||D_k^{(i)}(y,x) - D_k^{(i)}(G(x,s))||_1, \tag{5.5}$$

where $P$ denotes the number of discriminators and $D_k^{(i)}$ returns the output of the i-th layer with size $N_i$ of the k-th discriminator. The loss function can generally be interpreted as such that the generator should produce images that are very close to the real image as seen by the discriminator.

### 5.1.3 Image Stitching

Due to hardware limitations, the CGAN only allows predictions of $512 \times 512 \times 3$ with a common NVIDIA GPU that has 11 or 12 GB of memory. This issue needs to be addressed as the CGAN should be able to fully replace the camera in an MMS as shown in 4.1 which has a higher resolution of $2056 \times 2452$. When training the CGAN, image patches of size $1024 \times 1024$ are cut out of the real images $x$ and reduced to $512 \times 512$ and then passed to the CGAN. The method described below attempts to reverse this process. To generate full resolution images without re-training or modifying the CGAN architecture, a sliding window approach is used that generates many $512 \times 512$ sized image patches that are stitched together. The generator $G(\cdot)$ of the CGAN is moved over the original image $x$ having size $U \times V$ using a sliding window approach with step size of $k$ and a weight matrix $W$ with the size $1024 \times 1024$. Additionally there are two empty arrays $O$ and $N$ with size $U \times V$ (the original image size). The first one will contain the output image and the latter the weights in order to merge the patches. Also there exists a function $F(\cdot)$ that receives an image of size $1024 \times 1024$ and reduces it to $512 \times 512$. Conversely, $F^{-1}(\cdot)$ will do the opposite. Algorithm 5 shows how the CGAN can be used to create large images.

**Algorithm 5** Procedure to create a high resolution image

**Require:** Generator $G(\cdot)$, projected point cloud image $x$

1: **procedure** StichImage(x)
2: Initialise weight matrix $W$ with $W(n,m) = \frac{1}{2\pi\sigma^2} e^{-\frac{n^2+m^2}{2\cdot\sigma^2}}$
3: $O = array(U,V)$
4: $N = array(U,V)$
5: **for** i in range(0,U,k) **do**
6: **for** j in range(0,V,k) **do**
7: $O[i : i + 1024, j : j + 1024] \mathrel{+}= F^{-1}(G(F(x[i : i + 1024, j : j + 1024])))$
8: $N[i : i + 1024, j : j + 1024] \mathrel{+}= W$
9: stitched_image = $\frac{O}{N}$
10: return stitched_image

The algorithm first initializes a weight matrix $W$ with a Gaussian distribution, with its peak in the center of the matrix. Then the generator in line 7 is moved over the original image $x$ and the results are added to the matrix $O$ at the same position. In the same way the weight mask $W$ is added at the same position to matrix $N$. After both for loops are finished the full resolution image is obtained by calculating $\frac{O}{N}$.

## 5.2 Self-Supervised Shape Completion

The problem of self-occlusion in connection with this work occurs when an object is captured by a laser scanner and parts of the object block the view on the object itself. The problem has been described in Figure 4.2 where parts of the red car were not captured by the laser scanner. The problem is that ray tracing may not be able to cope with such occlusions if the camera and laser beams do not coincide. In the case of label transfer, 3D points behind a self-occluded object may be projected into the camera plane because they cannot be detected as occluded by the ray tracer. This leads to the effect of "label bleeding", where object labels are incorrectly assigned to points in the neighborhood of an object.

In this chapter, a GAN is presented that is able to learn the completion of objects in a self-supervised manner. A prerequisite for this method to work is a dataset of incomplete objects that belong to the same class that are roughly aligned. To be more precise, the GAN will be trained on a dataset of 3D point clouds containing incomplete shapes of the same class (e.g. cars). Because the GAN works only on grid-like structures the individual 3D point clouds are voxelized. The general approach is to divide each voxel grid containing an incomplete object sample into incomplete and complete subregions. The complete subregions are then used as a supervision signal so that the GAN can learn about completeness. The output of GAN is a voxel grid containing a complete shape that matches the incomplete input.

### 5.2.1 Subregion-Based GAN model

To learn how to generate complete shapes from incomplete ones, a GAN would need samples from the target distribution, i.e. complete shapes that define the positive "real" class of the discriminator. However, the assumption is that only incomplete shape observations are available. To gain access to a similar supervision signal, the discriminator is taught about completeness locally in a self-supervised manner by labeling complete subregions as "real" and incomplete regions as "fake" samples that should not belong to the target distribution.

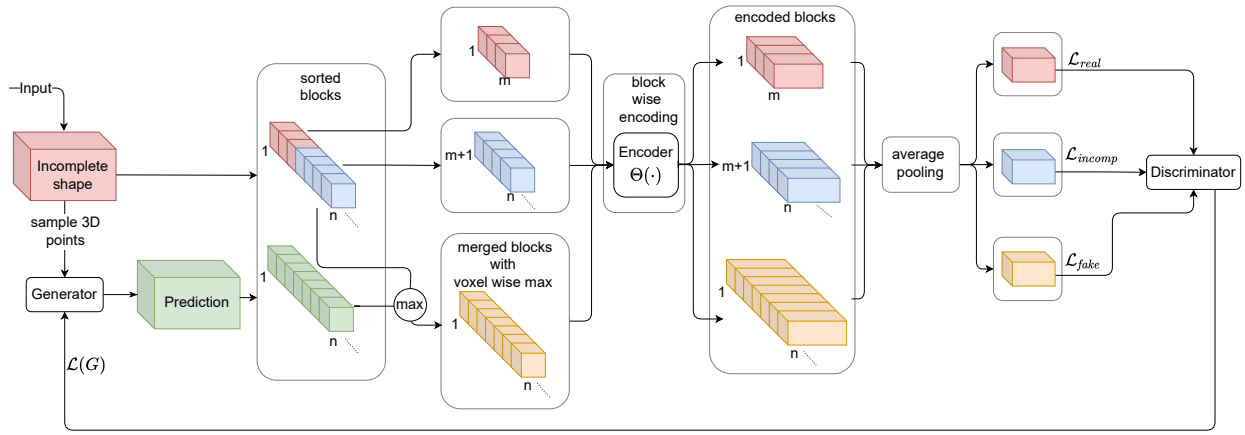



*Figure 5.4: Methodology: Red/blue and green color indicate real and generated data, respectively. Orange blocks are merged voxel-wise between real and synthesized, indicated by the "max" operation. The Encoder $\Theta(\cdot)$ will be described in the network architecture section. It encodes each subregion into a vector which are then average pooled to a fixed size representation and passed to the discriminator.*

This subregional GAN model is based on a binary voxel grid in which voxels on the 3D object surface are labelled as 1 (occupied) and all other voxels are labelled zero (free). Each voxel grid contains an object instance tiled into a fixed set of $n = n_x \times n_y \times n_z$ subvolumes called "blocks", see Fig. 5.4. These blocks are sorted by decreasing number of surface voxels in the real incomplete input. Under the weak assumption that blocks with high point density contain complete object parts, only a fixed number $m$ of the most densely populated blocks are passed to the discriminator as "real" examples, see red blocks in Fig. 5.4. The remaining $k = n - m$ less densely populated blocks from the real input are passed to the discriminator as "fake" examples, which are called "incomplete", see blue blocks in Fig. 5.4. This will ensure that regions with low point density are treated as fake, forcing the generator to output shapes with high point count. With the supervision provided so far, the discriminator can learn to ensure a sufficiently high point density, but it is neither biased against implausible shapes with appropriate point count, nor conditioned to the input.

The generator will be fed with the incomplete shape as voxel grid from which 3D points are sampled. These are then encoded into a feature vector using PointNet (Qi et al., 2017c) to ensure that the generator does not simply copy the input and learns to re-synthesise the incomplete shape. The prediction by the generator will also be tiled in a similar fashion as the real sample before and sorted by point density, see green blocks in Fig. 5.4. To ensure that the generated samples fits to the input, the discriminator is normally fed with a tuple containing a condition and the real or fake sample. Often the condition is the input of the generator which would also be given to the discriminator. However this cannot be done here, because the input and the real sample would be the same, which can be trivially detected by the discriminator. To solve this problem in each block, the set union between the surface voxels of the original input and those of the generator output is determined. Since 1 represents the surface labels and 0 the background, this corresponds to an element-wise max of the voxel labels, see yellow blocks in Fig. 5.4. The merged blocks will be passed to the discriminator as "fake" examples. By merging the input with the generated shape the discriminator should be able to detect if both shapes do not match.

The last problem to solve is that the "real", "incomplete", and "fake" blocks should not be discriminated individually, because this could prevent the individual parts in the blocks from not fitting together. Therefore each block is encoded individually by a series of 3D convolutions (Figure 5.6) into a fixed size feature vector of 256 each. This way three lists of $m \times 256$ "real", $k \times 256$

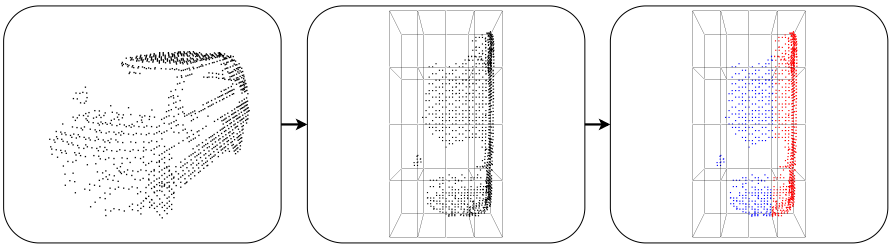


*Figure 5.5: First, a voxelized shape is taken as input (left image). Then, the input is divided into $n = n_x \times n_y \times n_z$ blocks (middle image in BEV). Each block is labeled as "real" (red) or "incomplete" (blue) based on the point density in each block.*

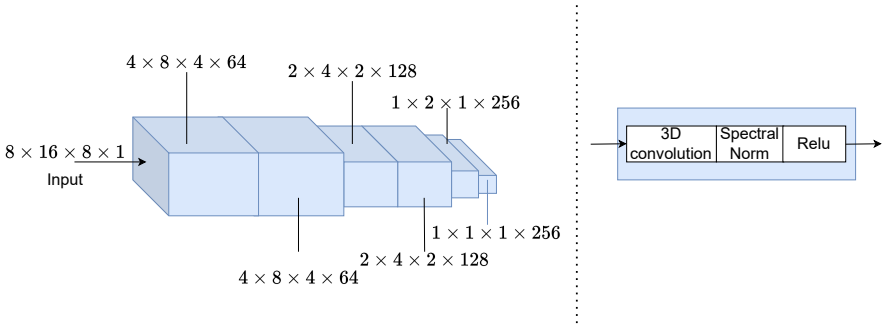



*Figure 5.6: This figure shows an example of network $\Theta(\cdot)$. It encodes a block of size $n_x = 8 \times n_y = 16 \times n_z = 8$ into a feature vector of size $1 \times 1 \times 1 \times 256$.*

"incomplete" and $n \times 256$ "fake" feature vectors are created, see Fig. 5.4 box "encoded blocks". Please note that the weights for the 3D convolutions are shared over all subregions. Now each list of feature vectors is average pooled to a fixed size of 256, see Fig. 5.4 box "average pooling". These averaged blocks are then each passed to two successive fully connected layers, which output the predictions of the discriminator for each subregion.

To sum up and to understand the intention behind the described model, consider a block that has a low point density in the generator output. In that case it is treated by the discriminator similar to an "incomplete" region, reinforcing the desired bias towards complete shapes. If the generated block has sufficiently many points, there are three cases: (a) the point count is correct, but the shape is implausible, leading the discriminator to assign a "fake" label and steering the generator towards plausible shapes. (b) The second case, the shape is plausible in itself, but not spatially aligned with the input, so that it will be assigned the "fake" label, which implements the conditioning on the input. (c) Finally the block matches the input and has the right density. If this case become frequent, the model has been successfully trained and the generator has learned to generate complete shapes that are well aligned with the input.

### 5.2.2 Loss Function

The GAN loss function in this section is derived from the least squares GAN-loss (LSGAN), Equation 5.1. Note that the subregion scheme can also be implemented using any other GAN-loss function, but LSGAN has been shown empirically to give the best results.

To describe the loss function, the following notation will be used. The real sample $x$ is split into blocks $q_j$ with $j \in \{1 \dots n\}$. Moreover, the ensemble of blocks sorted in descending order of point count is denoted by an asterisk $^*$ so that the sorted discriminator input becomes $q^*$ and the sorted

generator output becomes $G^*(x)$. The operation of encoding a block into a 1D vector with a series of 3D convolutions is denoted as $\Theta(\cdot)$. This function will represent the first part of the discriminator which is shown in 5.6. The same encoding for multiple blocks, followed by average pooling into a single 1D vector, is written as $\Xi(\cdot)$. This function represents the average pooled subregions in Figure 5.4 (box "average pooling"). tuples consisting

### Discriminator Loss

The discriminator loss has three terms, related to the description in the previous section. The first loss term is related to the $m$ most complete input blocks $\mathcal{L}_{real}(D)$. The second term $\mathcal{L}_{incomp}(D)$ relates to the $(n-m)$ remaining "incomplete" input blocks. The last term $\mathcal{L}_{fake}(D)$ is related to generated output blocks:

$$\mathcal{L}(D) = \mathcal{L}_{real}(D) + \mathcal{L}_{incomp}(D) + \mathcal{L}_{fake}(D) \tag{5.6}$$

As the target distribution should only contain complete shapes, the first term is simply the LSGAN loss for the "real" class,

$$\mathcal{L}_{real} = \Big(D(\Xi_{j=1}^{m}(q_j^*)) - 1\Big)^2, \tag{5.7}$$

where $D(\cdot)$ describes the discriminator network. To reject incomplete shape parts, the low-density blocks are assigned to the "fake" class, although they were not produced by the generator:

$$\mathcal{L}_{incomp} = \Big(D(\Xi_{j=m+1}^{n}(q_j^*))\Big)^2 \tag{5.8}$$

The third loss term is responsible for conditioning the CGAN on the input $x$. Unlike in classical CGANs such as pix2pix (Isola et al., 2017), there is no access to complete samples of the target distribution in this scheme. Usually, tuples consisting of an input and a generated sample or an input and a real sample would be passed to the discriminator to force the generator to respond appropriately to the input. To achieve a similar effect here, the generator output and the input shape are merged with the set union, see Section 5.2.1 and Fig. 5.4. By using 1 as the surface label and 0 as the background label, the set union can be written as an element-wise max operation as follows:

$$\mathcal{L}_{fake}(D) = D\Big(\Xi_{j=1}^{n}(\max(G_j^*(x), q_j^*))\Big)^2 \tag{5.9}$$

By using the set union, the discriminator is able to detect whether the generated sample matches the input or not and thus forces the generator to react appropriately for the input.

### Generator Loss

The loss $\mathcal{L}_{GAN}(G)$ is derived from Equation 5.1. The generator will be penalised for producing samples that are detected as fakes by the discriminator:

$$\mathcal{L}_{GAN}(G) = \Big(D(\Xi(\max(G_k^*(x), q_k^*)) - 1)^2 \tag{5.10}$$

Here all blocks $G_k^*(x)$ and $q_k^*$ are merged using the element wise max and the result is passed to the discriminator. Note that this loss also includes the "incomplete" blocks so that the generator is forced to produce subregions that have the correct point density and are plausible. The function $\Xi(\cdot)$ returns the averaged encoded subregions and passes them to the discriminator, which uses two fully-connected layers to output the estimate of whether the input is "fake" or "real".

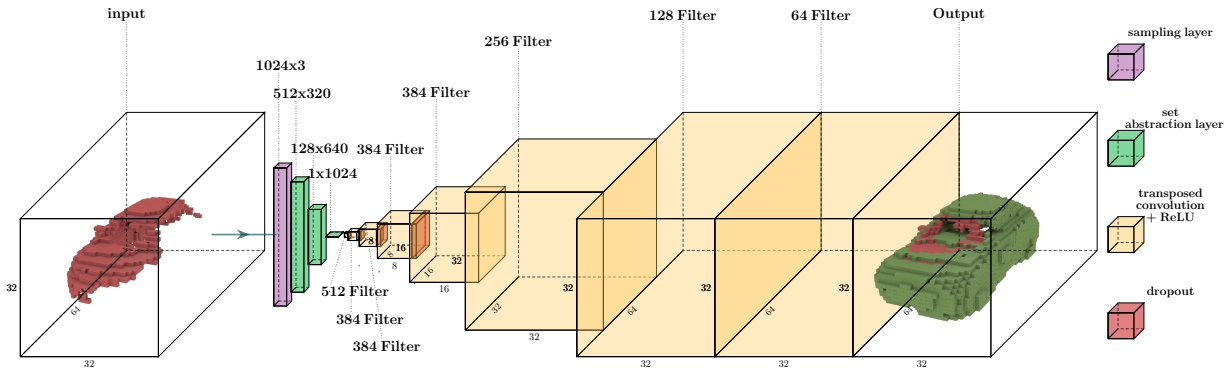



*Figure 5.7: The generator network receives an incomplete shape (red) as input and completes it. The network receives a voxel grid of size* $32 \times 64 \times 32$ *which is passed to a sampling layer (violet) that converts the grids to a point cloud. This point cloud is then passed to three PointNet++ set abstraction layers (green) that follow exactly the original implementation by Qi et al. (2017c) with the same hyperparameters. The encoded shape is then reshaped and passed to a series of 3D transposed convolutions (yellow) until the original size of* $32 \times 64 \times 32$ *is reached. The kernel size is* $3 \times 3$ *for the convolutional layer.*

Similar to pix2pix, the generator is guided by a least squares loss so that it favors surface points near the input points.

$$\mathcal{L}_{\ell 2}(G) = \frac{1}{m} \sum_{j=1}^{m} ||G_k^*(x) - q_k^*||_2 \tag{5.11}$$

Please note that the regularisation term is only applied to the first $m$ "complete" blocks so that the generator is not forced to produce subregions with low point density or incomplete subregions. Finally, a feature matching term is added that minimizes the difference between the encodings $\Theta(G_k^*(x))$ of the generated blocks with the encoding of the real blocks $\Theta(q_k^*)$.

$$\mathcal{L}_{feat}(G) = \frac{1}{m} \sum_{j=1}^{m} ||\Theta(q_k^*) - \Theta(G_k^*(x))||_2 \tag{5.12}$$

Similar to $\mathcal{L}_{\ell 2}(G)$ the L2-norm is calculated for the first $m$ "complete" blocks, again preventing the generator from being forced to produce incomplete or low density subregions. The complete loss function for the generator is created by minimizing the weighted sum of all three terms:

$$\mathcal{L}(G) = \mathcal{L}_{GAN}(G) + \alpha \mathcal{L}_{\ell 2}(G) + \beta \mathcal{L}_{feat}(G), \tag{5.13}$$

where $\alpha$ and $\beta$ are scalars to weight $\mathcal{L}_{\ell 2}(G)$ and $\mathcal{L}_{feat}(G)$.

### 5.2.3 Network Architecture

In the following, the architectures for the generator (Fig. 5.7) and the discriminator (Fig. 5.6) are defined. An obvious idea for the **generator** would be a straight-forward encoder-decoder structure with 3D convolutions and 3D transposed convolutions, as in 3D-GAN (Wu et al., 2016). Empirically, however, there are difficulties with such a design. It has been observed that skip connections can lead to local minima, where the generator just copies the incomplete input, rendering the whole GAN useless. To prevent the generator from doing that, not only the skip connections are removed, but the encoder part will use a completely different representation than the decoder. As can be seen in Figure 5.7 the encoder part of the network uses a point cloud representation and the decoder part is based on a voxel grid. Therefore, the **generator** uses PointNet++ set abstraction layers

(Qi et al., 2017c) to encode the input. The set abstraction layers were introduced briefly in Section 3.2 but are described in more detail in the following paragraph .

In order to encode the input shape, the voxel grid is first passed to a sampling layer that randomly selects a fixed number of 3D surface points, 1024 in this implementation. Note that the sampling layer has no trainable weights. The surface points are then passed through a series of PointNet++ set abstraction layers to finally obtain a feature vector of size $1\times1024$. Each set abstraction layer consists of three layers: (1) The sampling layer, which uses iterative farthest-point sampling to select a set of points which define the centroids of local regions for the following layer. (2) The grouping layer that constructs local sets by selecting neighboring points around the centroids of the local regions using a spherical query. (3) The PointNet layer, which uses a mini-PointNet on each group to learn local patterns and encode them into feature vectors. The generator uses a total of three set abstraction layers, see the green blocks in Fig. 5.7. The implementation and hyperparameters[1] are exactly the same as proposed by Qi et al. (2017c). The first set abstraction layer samples 512 out of 1024 points and outputs a feature vector of size 320 for each of these points, resulting in a $512 \times 320$ matrix where each row is associated with the original 3D point coordinates. The second set abstraction layer samples 128 points from this matrix and outputs a feature size of 640 for each of them, resulting in a $128 \times 640$ matrix. And finally, the third layer takes all the points and passes them to three fully connected layers with 256, 512, and 1024 units, respectively, resulting in a $128 \times 1024$ matrix. This matrix is then max pooled and reduced to output a $1 \times 1024$ feature vector describing the input. This vector is transformed into a $1\times1\times1024$ tensor and passed through the decoder of 3D transposed convolutions to obtain the prediction. To make the training more stable, batch-normalisation by Ioffe and Szegedy (2015) and spectral normalisation by Miyato et al. (2018) are applied at every layer. The activation function for all layers except the last one are ReLU functions. The output of the last layer is a sigmoid in order to give an estimate between zero and one whether an voxel should be marked as occupied or not.

The **discriminator** network is shown in Fig. 5.6. It consists of two parts: (1) The encoder $\Theta(\cdot)$ is a traditional sequence of 3D convolutions, again with spectral normalization in each layer. (2) The actual discriminator $D$, which receives a list of encoded blocks, calculates the average feature and classifies them as “real” or “fake”. For this two fully connected layers are used. The first with 64 output features and ReLU activation, the second with an output of one and no activation. In addition, no normalization is applied to the fully connected layers. During training, the encoder $\Theta$ is applied across all subregions and its weights are shared. The encoder and the fully-connected layers are trained end-to-end as part of the discriminator update.

[1] `https://github.com/charlesq34/pointnet2/blob/master/models/pointnet2_cls_msg.py`

# 6 Preparation of MMS data

In the previous chapters, the general methodology for label transfer was shown, including various extensions such as a correction scheme based on scanstrips and multi-view observations, and/or self-supervised methods to handle self- and dynamic occlusions.

In this section, the preprocessing steps and the description about the MMS database are presented, which is necessary to perform and understand the experiments in the next chapter.

## 6.1 Preprocessing of the Mobile Mapping Dataset

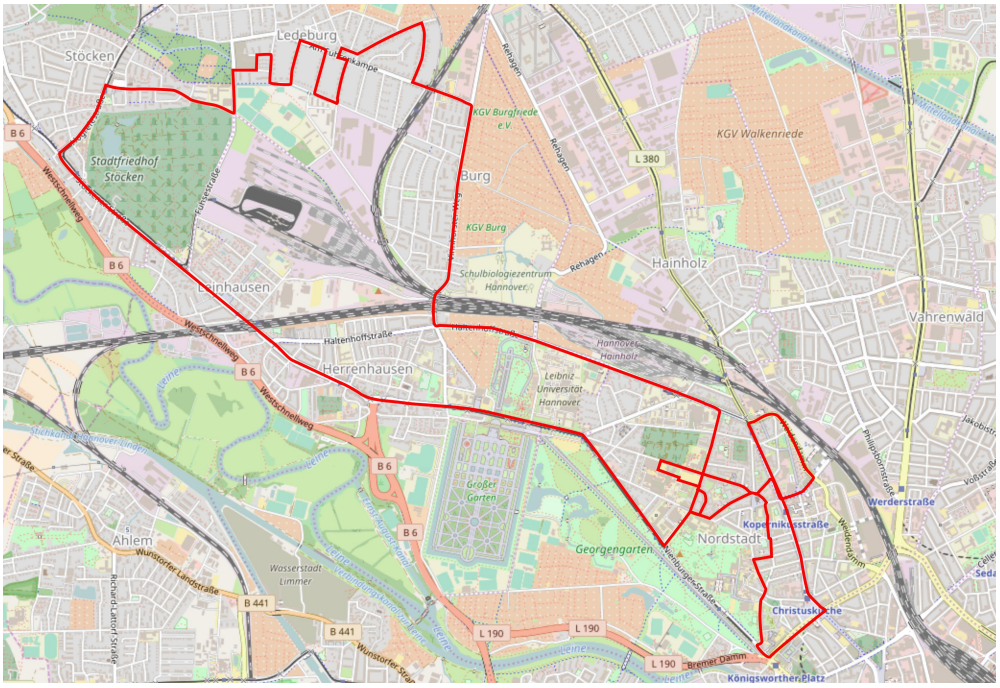



*Figure 6.1: Route of the MMS vehicle for the mapping campaign. The city center of Hannover is in the lower right corner and the Leibniz University is in the middle of the image.*

A fully calibrated VMX-250 mobile mapping system was used to acquire the data used in this thesis, s. Fig 2.3a. The MMS is equipped with two Riegl VQ-250 laser scanners, which have a maximum scan rate of 300 000 points per second with a range accuracy of ten millimeters each. In addition, two industrial cameras with a sensor size of $2056 \times 2452$ pixels each are used. For localization, the MMS has a high-precision GNSS/IMU system combined with a DMI. All trajectories are obtained by post-processing using reference data from the Satellite Positioning Service (SAPOS). The accuracy is within a lower decimeter range [1]

As shown in Chapter 4, the transfer process requires the generation of multi-temporal features such as the campaign count (Chapter 4.1.2) or photorealistic images (Chapter 5.1). Therefore, this work

[1] Riegl VMX-250 datasheet `http://www.riegl.com/uploads/tx_pxpriegldownloads/10_DataSheet_VMX-250_20-09-2012.pdf`

uses data created by Schachtschneider and Brenner (2020) from a bi-weekly long-term measurement campaign recorded in Hannover Germany, covering the same ∼20 km route over one year. The route includes areas of the city center as well as suburbs, multi-lane roads, parking lots and areas with heavy bicycle and pedestrian traffic. In total, the dataset contains 26 measurement campaigns covering different seasons with different weather and light conditions, as shown in Fig. 6.1.

*Table 6.1: Table with the dates of the individual measurement campaigns. The time of day shows that some started in the morning, others at noon or at dawn.*

| Year | 2017 | 2017 | 2017 | 2017 | 2017 | 2017 | 2017 | 2017 | 2017 | 2017 | 2017 | 2017 | 2017 | 2018 | 2018 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Date | 03/31 | 04/05 | 04/28 | 05/09 | 06/06 | 06/20 | 08/08 | 09/05 | 10/04 | 10/19 | 11/07 | 11/14 | 12/07 | 02/01 | 03/01 |
| From | 08:22 | 09:27 | 12:44 | 11:39 | 10:24 | 13:29 | 12:33 | 10:08 | 10:13 | 08:08 | 13:50 | 13:44 | 14:17 | 10:54 | 12:47 |
| To | 09:28 | 10:47 | 13:57 | 12:54 | 11:26 | 14:28 | 13:32 | 11:28 | 11:10 | 09:09 | 15:32 | 14:44 | 15:39 | 12:12 | 14:01 |

For this work, a subset of 15 of these measurement campaigns was used, see Table 6.1. This subset contains a total of 15 017 586 980 3D points and 236 380 images. In addition, the subset is fully aligned using the method described in (Brenner, 2016).

### 6.1.1 Semantic Segmentation of the MMS-Dataset

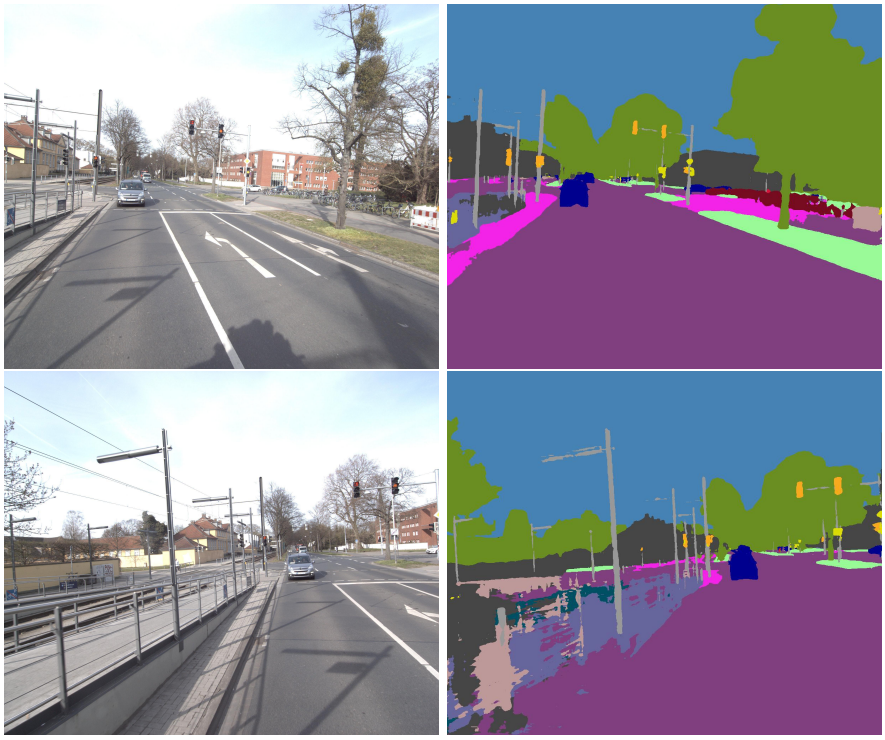


*Figure 6.2: Examples of semantically segmented MMS images of the same scene with two different viewing angles. The images on the left show the original MMS image. The images on the right show the corresponding predictions from Deeplabv3+. The MMS captures images pointing to the rear of the vehicle. The rows show the two available viewing angles.*

For the semantic segmentation of the MMS-images a DCNN is used that was pretrained on the Cityscapes dataset by (Cordts et al., 2016). The Cityscapes dataset offers 5000 highly accurate and additionally 20 000 coarsely annotated images from 50 different cities all over Germany. All images were taken in an urban environment by cameras mounted on a car and are pointing in the direction of travel. Each image is annotated pixelwise. There are 19 different classes, including

static objects such as buildings, streets, vegetation, street-signs and poles, and dynamic objects such as road users (cars, trucks, people, etc.). Each image has a size of $1024 \times 2048$ pixels.

The network used for semantic segmentation is Deeplabv3+ by Chen et al. (2018). At the time of publication, the network achieved an mIoU of 82.1% on the Cityscapes benchmark. The weights and source code were taken from the publicly available repository[2]. Since the images captured with the VMX-250 MMS have a size of $2056 \times 2452$ pixels, the network had to be adjusted accordingly to be able to make predictions on the higher-resolution images. Deeplabv3+ only takes whole images at once, therefore each image was resized to $1024 \times 1222$ before inference; a higher image size is not possible due to GPU-memory limitations. The Figure 6.2 shows two examples of semantically segmented images from the same scene with different viewing angles. It should become apparent that in both cases the network predictions delivers wrong predictions that will contribute to the label noise after the label transfer.

The pretrained network was used to semantically segment a total of 236 380 MMS images. As storing the full class distribution of the network would result in more than 1.18 terabytes of predictions for all images, these predictions are calculated on demand before each image is used for processing.

### 6.1.2 Human annotated MMS-Dataset

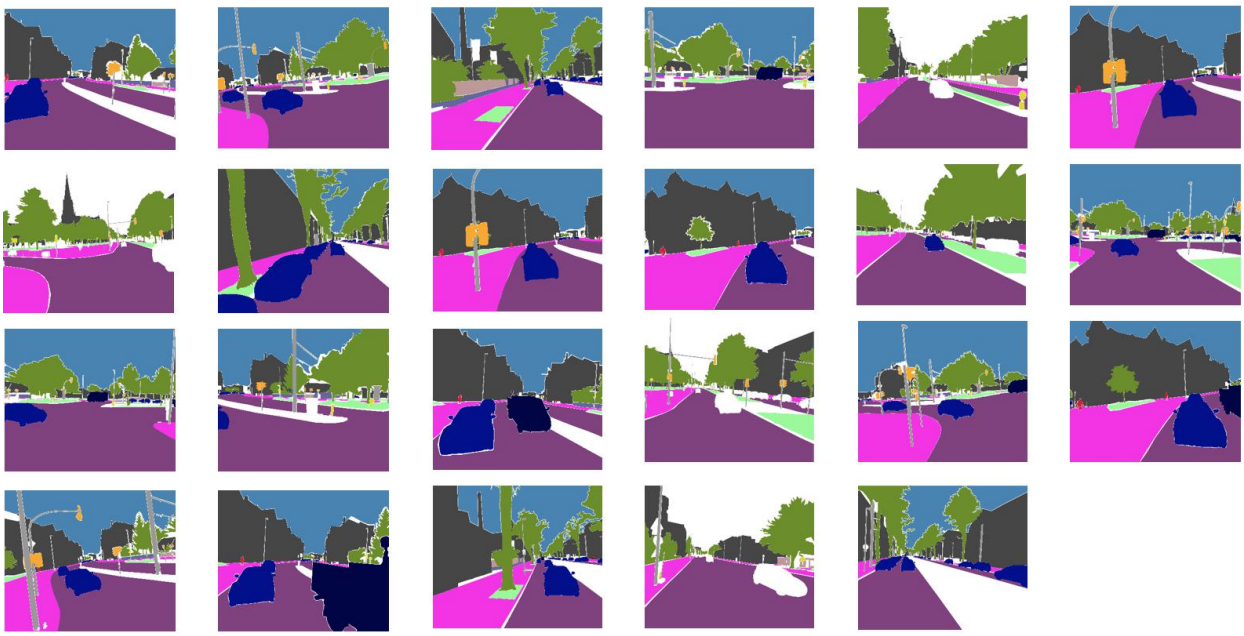


*Figure 6.3: 23 human annotated images. The original images are a subset of the MMS dataset. All images are labelled according to the official Cityscapes guidelines.*

To measure the performance of the pretrained Deeplabv3+ and for semi-supervised correction, a subset of 23 images was annotated according to the official Cityscapes label policy (Fig. 6.3). The images show three different scenes from different locations, they were selected such that they cover all classes that are used in this thesis. In addition, a small dataset of 10 images was created in which only bicycles were annotated.

[2] `https://github.com/tensorflow/models/tree/master/research/deeplab`

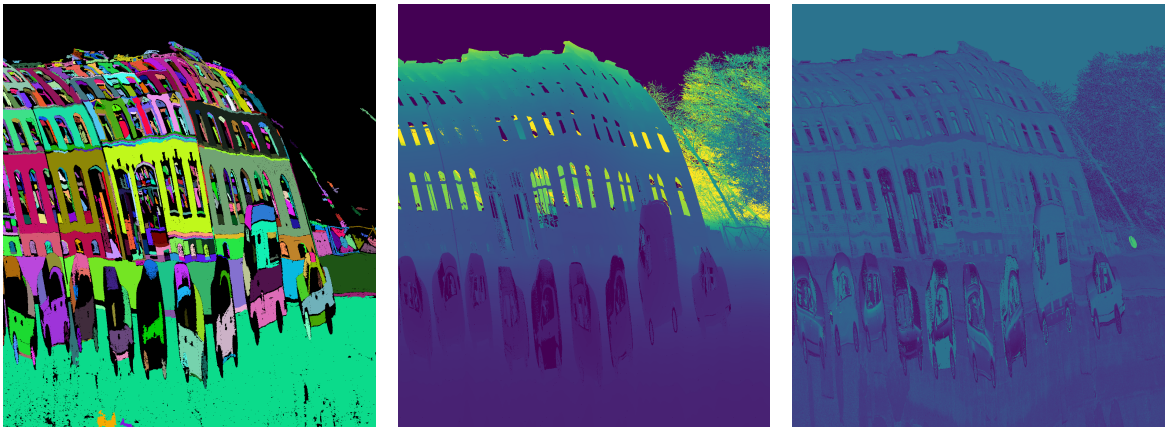
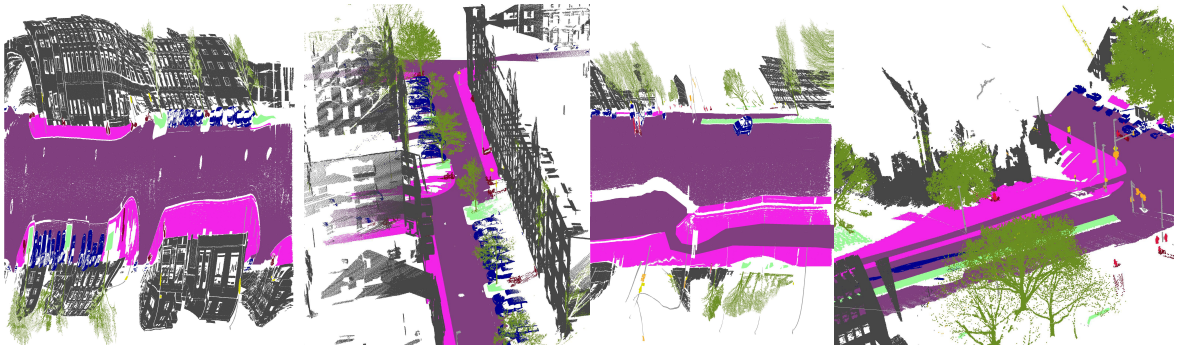


*Figure 6.4: Examples of different visualizations of the same scanstrip (images are cropped). Left: Results of the segmentation using the method described by Brenner (2016) (each segment randomly colored). Middle: Distance measured by the laserscanner. Right: Intensity measured by the laserscanner.*

To evaluate label transfer and learn how to correct wrong transferred labels, 14 scanstrips were manually annotated, resulting in 88 099 474 fully annotated 3D points. As this process is not straightforward, it is briefly described here:

First, each scanstrip is segmented using the method developed by Brenner (2016). In addition to segmentation, a layer of reflectance and range is added. All 3D points were annotated in the scanstrip representation using GIMP[3], an open source image editing program. Although the annotation process as described is faster and easier this way than direct annotation in 3D, it does add some label noise, especially due to "label bleeding". The reason for "label bleeding" is that it is difficult to hit the right pixels when painting in GIMP, especially with smaller objects, which can cause the label to bleed into the neighboring pixels of an object. To mitigate the problem, all scanstrips were edited three times by different persons. The first person created the initial labels, while the next two persons had to check and correct each annotation pixel by pixel. Some results are presented in Figure 6.5, which shows the final version of one of the annotated scanstrip and the corresponding visualization in 3D.



*(a)* Scene 1: Scanstrip *(b)* Scene 1: Point cloud *(c)* Scene 2: Scanstrip *(d)* Scene 2: Point cloud

*Figure 6.5: The images show examples of two labeled scenes. Each scene is shown once as annotated in the scanstrip (a and c) and as a 3D point cloud (b and d).*

[3] `www.gimp.org`

Table 6.2 shows the label distribution of all manually annotated data. The table is divided into two blocks, with the upper block containing all static classes and the lower block containing all dynamic classes. Also, the table shows that certain classes are not available, so the following simplification of Cityscapes class nomenclature is applied: *(i)* Class "sky" is excluded because there are no observations for it in the scanned 3D point cloud; and *(ii)* The classes "train", "rider", "cyclist", and "motorcycle" are excluded from training and testing because they are exceedingly rare in the annotated data.

| Name | Road | Sidewalk | Building | Wall | Fence | Pole | T.Light | T.Sign | Veg. | Terrain |
|---|---|---|---|---|---|---|---|---|---|---|
| Support 2D[%] | 32.7 | 8.5 | 16.7 | 0.3 | 0.3 | 1.3 | 0.3 | 0.08 | 13.6 | 1.4 |
| Support 3D[%] | 51.4 | 8.7 | 24.0 | 0.3 | 0.1 | 0.3 | 0.05 | 0.05 | 8.5 | 2.2 |
| Color | | | | | | | | | | |

| Name | Sky | Person | Rider | Car | Truck | Bus | Train | M.cycle | Bicycle | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| Support 2D[%] | 19.1 | 0.05 | - | 4.1 | 1.6 | - | - | - | 0.003 | 102M |
| Support 3D[%] | - | 0.07 | 0.005 | 3.2 | 0.5 | - | 0.3 | - | 0.2 | 88M |
| Color | | | | | | | | | | - |

*Table 6.2: The table shows the distribution of annotated classes in the images (Support 2D) and point clouds (Support 3D) of the MMS-dataset together with their corresponding color.*

## 6.2 Massively Parallel Point Cloud Rendering Using Hadoop

The CGAN presented in Section 5.1 must be trained with corresponding pairs of images from the source and target domains. As shown previously, the source domain contains projected point cloud images corresponding to the respective target image in the MMS dataset. The subset of the data used for training contains 15 billion (15 017 586 980) 3D points and 123 047 images. Due to pre-calibration, the intrinsic parameters of each camera are known. In addition, for each image, the position in the Universal Transversal Mercator (UTM), the orientation in roll, pitch, and yaw angles are given by the Riegl system.

To illustrate the amount of processing required to render all the images, naively, any of the 15 billion 3D points can potentially be projected into any of the 123 047 images, making the computational complexity problem bilinear, $\mathcal{O}(nm)$. If we assume for simplicity that both the number of 3D points and the number of images increase linearly with the size of the captured scene, it follows

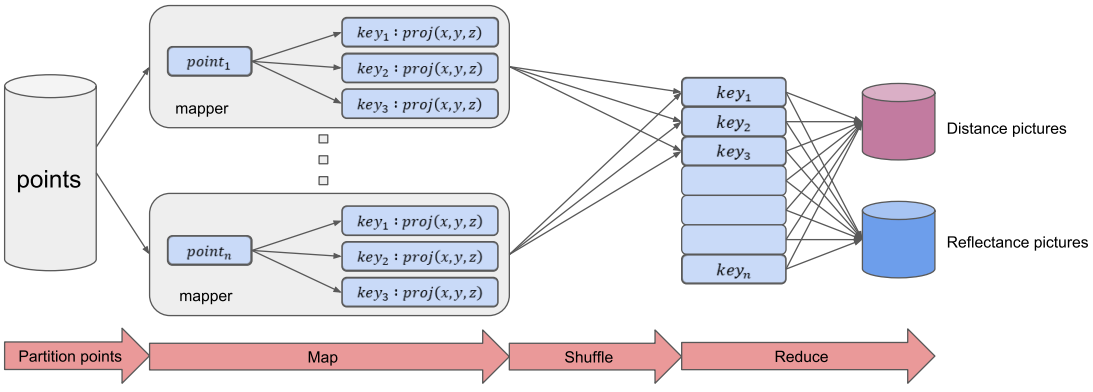



*Figure 6.6: The Scheme shows the MapReduce approach for rendering large point clouds. Image source (Peters and Brenner, 2020).*

that the computational effort is quadratic in the size of the scene. As each 3D point coordinate is stored as a 64-bit float in WGS 84 with the corresponding 32-bit intensity value, this means that about half a terabyte has to be processed, which is too much to keep all data in main memory. To solve this task, a massively parallel point cloud renderer was created on an Apache Hadoop cluster using the MapReduce framework, as shown in Fig. 6.6.

MapReduce is a framework for processing large datasets with parallel and distributed algorithms on a Hadoop cluster. In general, a mapper $M(k_1,v_1)$ receives a single key-value pair $(k_1,v_1)$ which will be processed. The mapper is applied in parallel to each pair in the input dataset. Each mapper may generate a list of new keys and values $list(k_2,v_2)$. After that the framework collects all pairs in the "shuffle" process with the same key $k_2$ from all lists and groups them by the key, resulting in the following key-value pair $(k_2,list(v_2))$. The reduce function $Reduce(k_2,list(v_2))$ is then applied in parallel to each key $k_2$, which in turn creates the final result.

The MapReduce principle is applied to the problem as shown in Fig. 6.6. In the first step, each mapper gets a single 3D point and its reflectance value. As each mapper has a list of all camera poses, a single point can be projected into each camera plane to check if the point is visible in a certain image. To avoid excessive calculations, the poses are filtered by the distance to the 3D point. Only poses that are within 300 m of a 3D point are retained for projection of that point. Additionally, as shown in the frustum check (Equation 2.6), poses where the point lies behind the camera are excluded. If the point passes both checks, it is projected into a camera plane. If the point lies inside the image, it will be included in the emitted key value list. Each mapping process possibly outputs up to one key value $list(k_2,v_2)$ pair per incoming 3D point for each image, depending on the number of images in which the point appears. The key $k_2$ is defined by the image name, the value $v_2$ contains the position of the projected point in the pixel coordinate system of the image $k_2$, the reflectance value and the distance between the camera center and the 3D point.



*Figure 6.7: Each row shows the same scene. The images on the left show the real camera images. The images in the middle show the projected point cloud colored by reflectance and the images on the right are colored by the distance.*

After all points are mapped, they are grouped per key $k_2$, resulting in a key-value pair ($k_2$,$list(v_2)$), see Figure. 6.6 "Shuffle". Each reducer receives one of these key-value pairs and loops over the list of values. As each value contains all necessary information to create the images, the reducer can insert the reflectance and distance values each into an image array. This produces two 16-bit gray value images per key, the first of which contains the distance and the second the reflectance values. However, if more than one point falls into the same 2D pixel, only value closest to the projection center is retained. Apart from that, occlusions are not considered. This means that objects in the front appear to be "transparent". Dynamic occlusions also occur regularly, which means that dynamic objects in the image may not match the objects in the projected images or vice versa. The process of rendering images for each mapping campaign needs about 8hrs to compute on a 6 server cluster with a total of 96 physical cores (each server has 16 physical and 16 virtual cores), which results in a total computation time of about 5 days. In addition to these images, an independent test dataset is created. It was recorded during a campaign in the city of Karlsruhe in February. The examples in Fig. 6.7, taken from Hannover, Germany, are intended to demonstrate some common problems due to neglecting occlusions.

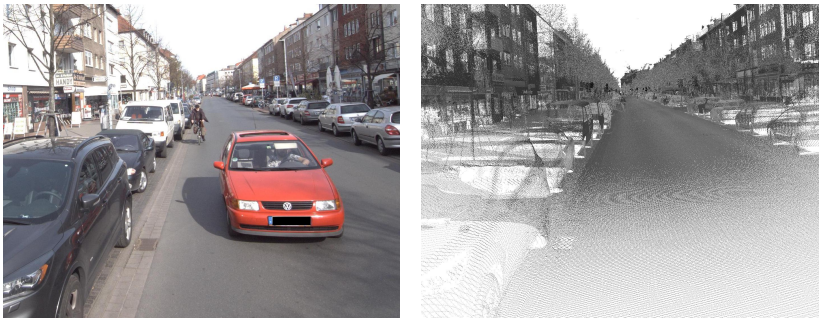


*Figure 6.8: Examples for dynamic occlusions. The red car and the bicycle rider appears in the camera image (left) but not in the point cloud (right).*

The upper scene in Fig. 6.7 shows a street with parked cars in front of buildings, the other scene shows a church. The rendered images appear very sparse the closer the 3D points are to the camera. In these areas, regular occlusions become more obvious. An example of this can be seen in the upper row, where the facades on the right side of the street appear transparent so that the buildings behind them are clearly visible. Dynamic occlusions also occur regularly. For example, the red car and the bicycle in Figure 6.8 are visible in the camera image, but not in the point cloud.

## 6.3 Datasets of Self-Occluded Objects

This section shows how the datasets for the method in Chapter 5.2 were created. The goal is to create databases with self-occluded objects that are globally aligned. One object in the MMS dataset that suffers frequently from self-occlusion is the class *car*. Due to the low angle of view from the road, cars are typically occluded 40-80% of the time in the scanned point cloud, with at least one side missing completely in most cases. Since no real dataset with occluded cars is available, the cars were extracted from the mobile mapping system data shown in the previous section. However, for these extracted cars, there are no reference point clouds of complete cars, which makes it difficult to measure whether they were successfully completed. Therefore, the Section 6.3.2 shows how to create synthetic datasets of occluded objects for which a complete shape (ground truth) exists.

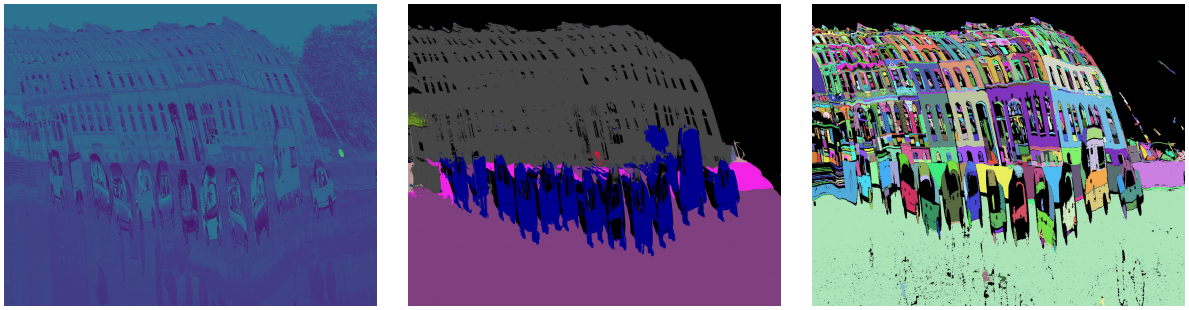
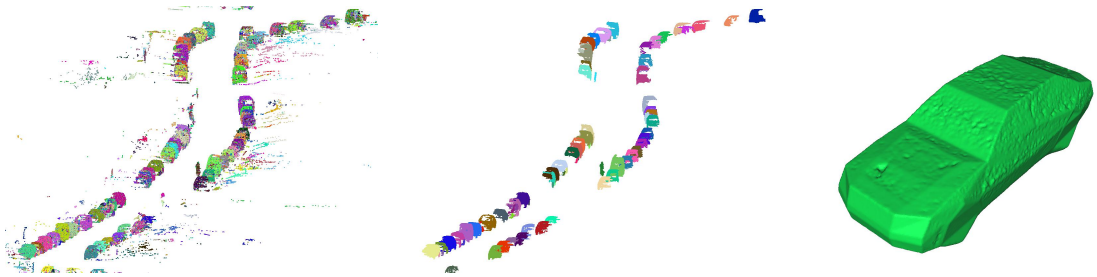


*Figure 6.9: The same scanstrip scene colored by reflectance (left), class labels (center) and segment ID (right).*

### 6.3.1 Real Dataset

The aim of the following procedure is to instantiate and align cars from the scanned MMS point clouds. In general, the procedure works by removing all non-car points from the dataset, followed by an instantiation step, a filter step to remove outliers, and finally a global alignment of all cars. In the first step all 3D points are semantically segmented. For this the naive label transfer as shown in Chapter 4.1.3 is used. In the second step, all points that are not assigned to the *car* class are removed. Due to the presented problems of naive label transfer, the environments of the cars are frequently assigned to the class *car*. Such points must be eliminated in a filtering step. The problem can be visualized very well in the scanstrip representation of the point cloud as shown in Figure 6.9b (middle). The image clearly shows that the *car* labels (blue) are sometimes wrongly assigned to surrounding areas. The scanstrip on the left (6.9a) can be used for comparison. In order to differentiate between surrounding areas and car areas, the scanstrips were segmented using the graph-based image segmentation by Brenner (2016), Fig. 6.9c (right). As surrounding areas of cars are often sidewalks, facades or streets, they often form very big segments in the segmented scanstrip. In order to remove points which are wrongly assigned to the *car* class, the number of points per segment are counted. If the minority of points per segment belong to the *car* class, they are removed, otherwise they are kept.



*(a)* extracted cars containing noisy labels

*(b)* extracted cars after filtering

*(c)* prototype car

*Figure 6.10: 3D street scene which shows only points classified as car, colored randomly by instance (before and after filtering of outliers) and a prototype car used for orienting incomplete scans (right).*

The final step is to instantiate all cars in the filtered point cloud to form the dataset of individual car samples. In order to do this, region growing based on the estimated surface normals is used. As described in 2.1.2.3, region growing uses seed areas which are expanded to their neighbouring points. Because region growing is susceptible to leakage, which means that a region can bleed into surrounding cars (Fig. 6.10a), an additional filter step was added. First the the minimum enclosing

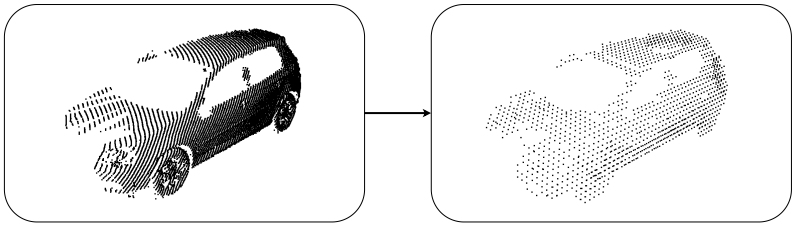


*Figure 6.11: Example of an extracted car before (left) and after voxelization (right). The voxelized point cloud shows only the center of each voxel*

rectangle for each car instance was calculated using the convex hull of proposed car instance. Then a car candidate was kept if the area of the rectangle was between 5 and 16 $m^2$ and if the main axis was shorter than 6 $m$. The overall result can be seen in Figure 6.10b. Next all cars are aligned as such that they have all roughly the same orientation and position. First, the front of the vehicle is determined by comparing the height differences along the main axis. From experience, the highest points belong to the rear part of the vehicle. Due to the scan geometry, most cars are scanned from the side, making this method more successful. Afterwards, all incomplete scans are aligned to a generic car template (Fig. 6.10c) using ICP.

After the alignment all cars are voxelized. For this a grid with volume of size $32 \times 64 \times 32$ is used. Each car is placed so that the major and minor axis of the car are parallel to the y and x axis of the voxel grid. The barycenter of the car lies roughly in the middle of the volume at $x = 15$, $y = 32$, $z = 8$. In order to fit each car to the grid, they are scaled by a factor of 0.75. Due to the relatively low voxelgrid size the cars have a low resolution, but are still recognizable, s. Fig. 6.11. For comparison, a dataset with different cars and different characteristics was created. The KITTI dataset from (Geiger et al., 2013) was used for this purpose. It contains labeled point clouds of Velodyne HDL-64E scans. Each car in this dataset is annotated with a 3D bounding box. Based on these labels, all cars could be extracted from the dataset. Finally, all cars were aligned and voxelized in the same way as the extracted cars from the MMS dataset.

### 6.3.2 Synthetic Datasets

The aim of the following procedure is to create synthetic datasets of occluded objects for which a complete shape is available as reference. These *synthetic* datasets are based on Shapenet by (Chang et al., 2015) and Modelnet by (Wu et al., 2015). Both datasets consist of several object categories like airplanes, cars or even furniture. All objects are aligned and stored as polygon meshes, which are collections of vertices, edges and faces that define the polyhedral object surfaces. The task is to create self-occluded samples of each object to test how well the missing surface can be recovered. The reference will be given by the complete watertight objects in the dataset itself. To create occluded samples, a similar procedure as described by Stutz and Geiger (2018) is used.

Self-occlusion could be simulated by randomly cutting off parts of each object, thus fulfilling the requirement that each object is "incomplete". However, to simulate realistic self-occlusions, the objects should be incomplete wherever they are occluded by themselves when measured with a sensor. This is done by randomly rotating the meshes to sample different viewing angles and then projecting them onto a 2D grid to obtain points using Equation 2.4. Since the 2D grid can only store non-occluded surface points, self-occlusion is simulated by back-projecting the points into a



*Figure 6.12: Examples of self-occluded and voxelized objects from the Shapenet dataset. Left: A complete car and the result of the simulated self-occlusion. Right: A plane and its self-occluded counterpart. All voxels are colored by height, where green is low and red is high.*

3D voxel grid. In order to create the synthetic datasets from ShapeNet and ModelNet the intrinsic camera matrix is defined as:

$$P_{modelnet} = \begin{bmatrix} 96 & 0 & 32 \\ 0 & 96 & 32 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

For ModelNet an image size of $64 \times 64$ is used. The objects are voxelized into a $32 \times 32 \times 32$ grid. By adjusting the virtual camera parameters the sampling density can be controlled. The density is measured by the mIoU between the incomplete and the complete shape in the voxel grid. To test densities in ShapeNet two different camera matrices are defined, one for high resolution ($P_{High}$) and one for the low resolution samples ($P_{Low}$). The image size for the high resolution dataset is $640 \times 640$ pixels, and for the low resolution it is $64 \times 48$ pixels. Both datasets were voxelized into a $32 \times 64 \times 64$ grid.

$$P_{Low} = \begin{bmatrix} 96 & 0 & 32 \\ 0 & 120 & 24 \\ 0 & 0 & 1 \end{bmatrix} \qquad P_{High} = \begin{bmatrix} 860 & 0 & 320 \\ 0 & 860 & 320 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.2}$$

For *planes* the densities[4] are $\approx 0.49$ mIoU ($P_{High}$) and $\approx 0.12$ mIoU ($P_{Low}$). For Shapenet *cars* they are $\approx 0.09$ mIoU ($P_{High}$) and $\approx 0.024$ mIoU ($P_{Low}$).

The extrinsic camera parameters for the ModelNet shapes are defined as follows: The mesh is first rotated using a rotation matrix $R_m = R_x(\alpha_m)R_y(\beta_m)$, where $\alpha_m \in [-60°,0°]$ and $\beta_m \in [-180°,180°]$ are randomly chosen, they define the amount of rotation about the x and y-axes, respectively. The mesh is then translated by a fixed translation vector $T_m = [0,0,2]^T$. For ShapeNet the meshes are rotated using the rotation matrix $R_s = R_x(\alpha_s)R_y(\beta_s)R_z(\gamma_s)$ with $\alpha_s \in [-45°,45°]$ and $\beta_s \in [-180°,180°]$ and $\gamma_s \in [-45°,45°]$. The translation of the mesh is done with the fixed translation vector $T_s = [0,0,2]^T$. From every object, a number of 10 self-occluded voxel-grids are sampled. Exemplary results for Shapenet are shown in Figure 6.12.

[4]The density was measured against the plane hull and not the filled shape

# 7 Experiments and Results for Multi-View Label Transfer

## 7.1 Introduction

The chapter is structured as follows: First a baseline for label transfer is presented. The baseline is examined in detail with regard to the introduction of label noise in 3D. Furthermore, different noise sources are identified using 2D and 3D reference data. Next the training, validation and test data is introduced that is used in most experiments. If experiments use a different dataset it will be shown in the corresponding chapter. The experiments in the following chapters will show that handling these errors reduces the label noise compared to the baseline. Each method presented in Chapters 4 and 5 will be examined in detail using the data presented in Chapter 6.

Table 7.1 gives an overview of the presented methods and the identified errors that affect the label noise. It is indented to show qualitatively the strengths and weaknesses of the methods. The baseline is given by the naive label transfer presented in Chapter 4.1.

*Table 7.1: The table shows the identified types of errors (rows) and the introduced methods (columns) for label transfer. It gives a qualitative overview of which method treats which type of error. The symbols in the table are as follows: "-" is not considered, "o" is treated implicitly and "+" is considered explicitly.*

| Error \ Method | Naive (4.1) | Point-wise (4.2.1) | SNet (4.2.1) | MVNet (4.3) | LTNet (4.3.2) | Point Cloud Rendering (5.1) | Shape Completion (5.2) |
|---|---|---|---|---|---|---|---|
| Regular Occlusion | + | + | + | - | + | + | - |
| Label-Policy Errors | - | o | o | - | + | - | - |
| Calibration Errors | - | o | o | - | + | + | - |
| Prediction Errors | o | o | o | + | + | - | - |
| Dynamic Occlusion | - | o | o | - | + | + | - |
| Self-Occlusions | - | o | o | - | + | o | + |

Regarding the training, testing and validation data for the experiments: In the following section, the baseline is measured using the **entire** dataset presented in Section 6.1.2. Based on these results, a training set, a validation set, and a test set are presented in Section 7.3. These datasets are used for all methods except MVNet, which uses a different dataset presented in the respective chapter.

## 7.2 Baseline

The (naive) baseline is given by the label transfer from a 2D image classified by a pretrained DCNN to a 3D point by exploiting the geometric correspondence between camera and laser beam in a fully calibrated system. The best case would be that the DCNN makes no mistakes and that the 2D pixels and 3D point cloud points are perfectly matched and point to exactly the same point in the real world. As this is not always the case, some points in 3D are assigned a wrong class label that does not match the real object they represent. To mitigate these problems, two methods have been introduced. First, ray tracing is used to find occluded 3D points that do not coincide with the camera rays – these points are discarded during the label transfer. Secondly, the label transfer is done using multi-view predictions, which are aggregated in a histogram $h$ for a single 3D point. If the DCNN makes few wrong predictions, they can be removed by assigning the class label to the 3D point that has received the most votes in the histogram (majority vote).



*(a)* Input *(b)* Deeplabv3+ *(c)* Ground Truth

*Figure 7.1: An example which shows the prediction quality of Deeplabv3+ on the MMS-images. The Figure shows the input (left), the semantically segmented image (middle) and the corresponding ground truth image (right).*

The baseline performance is measured by comparing the predictions of a DCNN pretrained on Cityscapes to the human annotated MMS images (see Section 6.1.2). Since the pretrained network was never trained on this dataset, it is likely that the performance will deteriorate compared to the Cityscapes test set; this is called *domain gap*. In a second step, the predictions are applied to the 3D points using the naive approach, as shown in Chapter 5. Here all 3D points are labelled by the majority vote from the multi-view predictions. Finally, the transferred labels are compared with the human-annotated 3D reference set (see Section 6.1.2), which yields the amount of wrong class assignments in 3D. By comparing the quality before the label transfer (in 2D) and after the label transfer (in 3D) and by investigating the confusion matrices, different noise sources and types are identified.

### Results and Evaluation

To assess the performance of the pretrained DCNN on the MMS-dataset the Jaccard Index is used which is also known as the Intersection over Union (IoU).

$$\mathrm{IoU} = \frac{TP_i}{TP_i + FP_i + FN_i} \tag{7.1}$$

$\mathrm{TP}_i$ $\mathrm{FP}_i$ and $\mathrm{FN}_i$ are the numbers of true positive, false positive, and false negative pixels, respectively, determined over the whole test set for each class $i$. The mIoU is then calculated by the unweighted mean over all classes. For the MMS-dataset the intersection over union $\mathrm{IoU}_{2D}$ is calculated between the 23 annotated ground truth images $y_{2d}$ and the corresponding predicted images $\hat{y}_{2d}$, see Fig. 6.3. The estimated domain gap $\epsilon_{2d}(i)$ per class $i$ is then defined as the distance between $\mathrm{IoU}_{2D}(i)$ and the intersection over union $\mathrm{IoU}_{city}(i)$ on the original Cityscapes test-set the DCNN was trained on.

$$\epsilon_{2d}(i) = \mathrm{IoU}_{city}(i) - \mathrm{IoU}_{2d}(i) \tag{7.2}$$

The quality of the prediction can be seen in the images in Figure 7.1. The example shows that Deeplabv3+ performs well for the classes *pole* and *terrain*, which are detected accurately. However, the sidewalk is recognized very poorly, which can be seen in the lower left part of the image. The results for **Deeplabv3+** (Fig. 7.2) are only shown for classes which annotations are present in both the 2D images and the 3D point cloud. The blue bars show the IoU per class on the Cityscapes dataset, the orange bars show the result for the same model on the MMS dataset. Deeplabv3+ has a significant performance degradation on the MMS data. The total mIoU decreased from

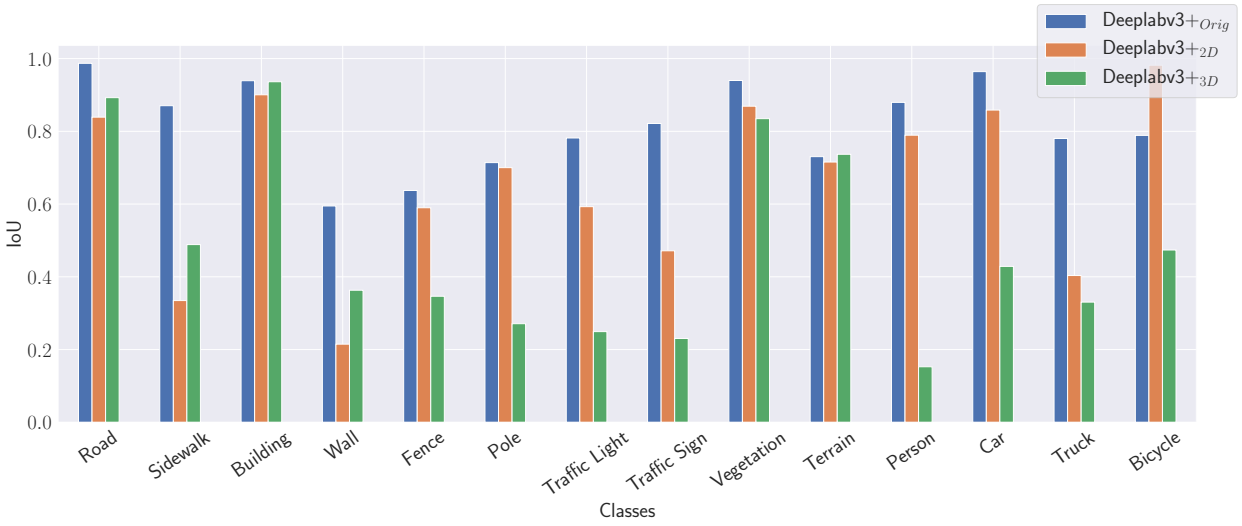



*Figure 7.2: Deeplabv3+ baseline: The figure shows the IoU per class on Cityscapes (Deeplabv3+$_{Orig}$), on the MMS-images (Deeplabv3+$_{2D}$) and after the naive label transfer into 3D (Deeplabv3+$_{3D}$).*

0.818 to 0.661. The largest performance degradation is seen in the *sidewalk*, *wall*, *truck* and *traffic sign* classes, which decreased by an $\epsilon_{2d}$ of 0.53, 0.38, 0.38 and 0.35. On the other hand, classes such as *pole* and *terrain* only suffer from an error of $\epsilon_{2d}(\text{pole}) = 0.014$ and $\epsilon_{2d}(\text{terrain}) = 0.015$. Interestingly IoU for *bicycle* even increased on the MMS-dataset by $\epsilon_{2d}(\text{bicycle}) = -0.19$. One possible reason is that the annotation process differs in the two reference datasets. For example, due to the efficient labeling, only bicycles in the foreground of the images are annotated, which may ease their detection by the DCNN.

Next, all predictions from the MMS images were mapped to the corresponding 3D points using the method described in Section 4.1. For the mapping, regular and self-occlusions are compensated by ray tracing as described in 4.1. Other types of errors, such as dynamic occlusions, prediction errors and calibration errors are not considered. Since there can be any number of 2D predictions for a 3D point, the final class is decided by majority vote over all 2D predictions. If there is no majority, the 3D point is removed from evaluation. Finally, the 3D points which are annotated this way, are evaluated using the human annotated reference set (Chapter 6.1.1).

The performance of the naive label transfer is estimated by calculating the IoU between the transferred labels $\hat{y}_{3d}$ and the reference set $y_{3d}$, see Table 6.2. The green bars in Figure 7.2 show the IoU per class. By mapping the labels into the 3D point cloud the mean IoU decreased significantly from 0.638 (orange) to 0.481 (green). A closer inspection shows that not all classes decreased in performance after the mapping. The classes *sidewalk*, *wall*, *road*, *building* and *terrain* have improved. A likely reason for this is that some wrong predictions could have been removed by the majority vote. To get a better understanding of what kind of error is introduced by label transfer, two confusion matrices are shown in Figure 7.3. The matrix on the left shows the results for Deeplabv3+ on the 2D MMS dataset, i.e. **before** the labels are transferred. The other matrix shows the results **after** the mapping process in 3D. Theoretically, there should be no difference between the two confusion matrices if the camera- and laser-beams are perfectly coincident and always point to the same object. As this is not the case, the differences between the two matrices include mostly the errors that arise from the transfer of the labels (errors that arise from human annotation are neglected). As a side note the matrices are normalized by row, because the annotations are very imbalanced they don't show to which extend the classification suffers by a wrong prediction.

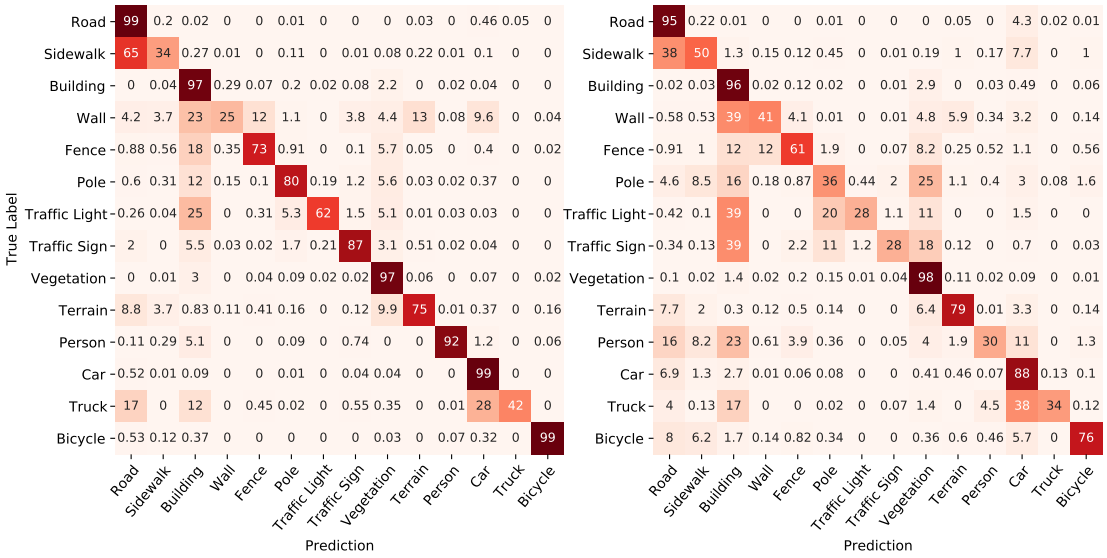



*Figure 7.3: Confusion matrices for Deeplabv3+ before (left) and after mapping into 3D (right). Both matrices are normalized per row.*

The **left matrix** shows the confusion matrix derived using the 2D images: At a first glance it should become clear that the classes *sidewalk*, *wall*, *truck* are the weakest, but for different reasons. Looking at the second row, the class *sidewalk* is strongly confused with the class *road*, which can be considered a "semantic problem", similar to the penultimate row, where *truck* is very often predicted as *car*. This means that both classes are semantically very close to each other. The class *wall* is different, it is confused with almost all other classes.

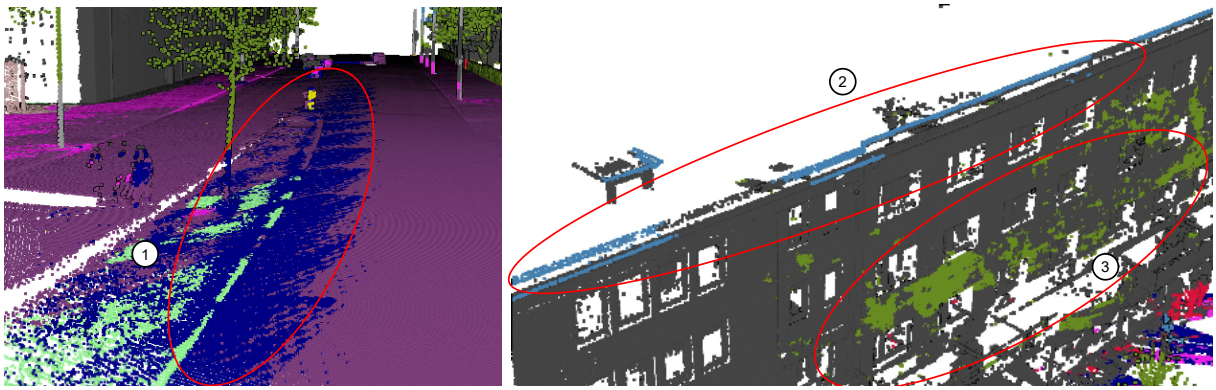



*Figure 7.4: Examples for different types of labelling errors: The first image shows dynamic occlusions (Ellipse 1). The Second image shows calibration errors (Ellipse 2) and Label-Policy errors (Ellipse 3)*

The confusion **matrix on the right** shows very different errors. The following reasons are suspected as causes:

– **Calibration Errors**: This problem may occur with class types that have a small or thin physical size, such as *pole*, *traffic light* or *traffic sign*. Although these classes were well predicted in 2D, they are very noisy after label transfer. For example, they are all regularly confused with classes that surround these objects in images, such as *building*, *sidewalk*, *pole* and *vegetation*. Figure 7.4 ellipse 2 shows an example where parts of the building (gray) are assigned *sky* labels (light blue), which is clearly a calibration error since it is impossible to measure the sky with a LiDAR.

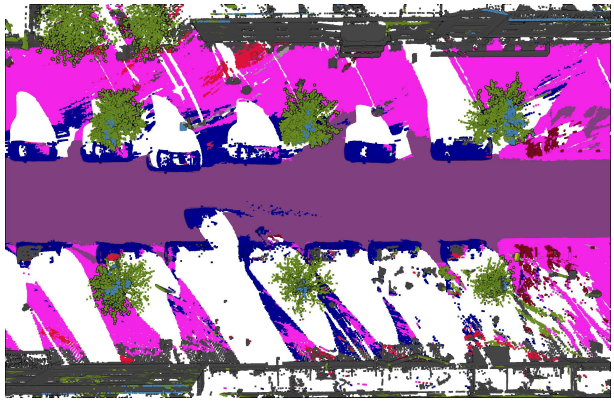


*Figure 7.5: An Example for self-occluded cars (blue), as seen from above.*

- **Dynamic occlusions**: All dynamic classes such as *person*, *car*, *truck* and *bicycle* could fall into this category as long as they are moving during the recording of the data. As seen in the right confusion matrix these classes are regularly confused with *road*, *sidewalk* and *building*. Figure 7.4 shows an example where a road (violet) is wrongly assigned to class *car* (blue). The reason for this should be that the annotated images captured a moving car that was not captured by the laser scanner thus leading to a wrong label transfer.
- **Label-Policy Errors**: Labeling-policy errors as already described in 4.6 can be seen in the third row of the right matrix. After the transfer, the class *building* is more often confused with *vegetation*. Fig. 7.4 ellipse 3 shows an example where the building (gray) is sometimes assigned to the class *vegetation* (green).
- **Regular occlusions and self-occlusions**: These types of errors are difficult to separate from the others, especially from calibration errors. Even if they are mitigated by ray tracing, they still occur. Regular occlusions can be seen in objects that are in the background in a typical image, such as buildings. In the third row of the right matrix, some classes are more often confused with *building* than before label transfer. Self-occlusion occurs when, for example, a car occludes parts of itself to the laser scanner, while these parts are visible in the images. The result is that the *car* labels are mapped onto the sidewalk, which can be seen in the second row of the right confusion matrix, see Fig. 7.3. Figure 7.5 shows a street scene in which the *car* labels (blue) bleed into the sidewalk (bright pink). As these cars are parked this error is not due to dynamic occlusion.

## 7.3 Training, Validation and Test Set

In total, there are 14 human annotated scanstrips, see Section 6.1.2. The mIoU for the baseline was estimated using the entire reference set. However to train a classifier for label error correction, the reference set must be divided into a training, validation and testing subset. As the reference set is highly imbalanced and relatively small, a division is not trivial and must be done very carefully. First, three requirements are defined that should be met by each subset so that a meaningful evaluation can be made and both underfitting and overfitting is prevented.

1. The subsets should be separated spatially. Empirically, a classifier trained on 3D data tends to overfit to the test set if it is selected randomly. This is because nearby 3D points often contain very similar features and labels.
2. All subsets should cover all supported classes.
3. Each subset should contain mostly "representative samples" so that they have a very similar label error distribution as the entire dataset. At best, the training, validation, and testing set each have the same IoU per class as the entire dataset in Figure 7.2. If a classifier performs well, it should be easy to compare the performance on the test set with the estimated noise level on the whole reference set.

To satisfy the first requirement, the subsets are divided by the scanstrips, each of which contains a locally separated scene and does not overlap. The next task is to find two subsets, one of which should contain the training data and the other the testing data. The first subset is later divided into training and validation data.

In order to find representative subsets, the IoU per class $i \in \{1, \cdots, 19\}$ for the training set $\mathrm{IoU}_{\mathrm{train}_k}(i)$ and test set $\mathrm{IoU}_{\mathrm{test}_k}(i)$ is estimated. There are $k \in \{1, \cdots, 5887\}$ combinations for the 14 scanstrips as such, so training set and the testing set support all classes. To fulfill the second and third requirement, the mean distance $\delta_\epsilon(k)$ between the IoU for subset $k$ and the estimated IoU for $\mathrm{IoU}_{3D}$ in Chapter 7.2 should be minimal:

$$\begin{aligned}\delta_\epsilon(k) = \frac{1}{2n}\Bigg( &\sum_{i=1}^{n} |\mathrm{IoU}_{\mathrm{train}_k}(i) - \mathrm{IoU}_{3D}(i)| \\ &+ \sum_{i=1}^{n} |\mathrm{IoU}_{\mathrm{test}_k}(i) - \mathrm{IoU}_{3D}(i)| \Bigg)\end{aligned} \tag{7.3}$$

The minimal value of $\delta_\epsilon(k)$ is found by iterating over all combinations. The best split has a distance of 0.027 which means that the mean difference between the estimated amount of label error in the training set and testing set deviate only by 2.7% from the one of the entire reference set. Based on the total number of points the reference set is divided into 74% training (9 scanstrips) and 26% test set (5 scanstrips) which is a typical split. Depending on the type of correction (point-wise or scanstrip-based), the validation set is determined differently. The test set will be always the same in the following subsections, so that all methods can be compared with each other.

## 7.4 Scanstrip-Based Correction

In this section the methods for point-wise correction (Section 4.2.1) and scanstrip-based correction (Section 4.2.1) are evaluated. They try to correct the labels on the 3D points, after the label transfer. The methods are therefore evaluated on the 3D point cloud reference set (Chapter 7.3).

### 7.4.1 Point-Wise Correction

This section introduces the baseline for label noise correction. The correction is based on point-wise features to show how these features affect the correction process and what a fine-tuned (classical) classifier is able to achieve. A gradient boosted decision tree is used for this purpose. For a more efficient use of the data, the classifier is trained with 3-fold stratified cross-validation. In contrast to normal cross-validation, where random subsamples are taken, stratified cross-validation maintains the percentage of samples for each class. The best parameters are found by maximizing the mIoU using a two-step grid search.

In the first step, the learning rate is set to 0.3. In addition, a sub-sample (or bagging fraction) of 0.8 is used. This means that only 80% of the rows from the training set are used to fit each tree. The number of leaves is fixed to 80. The number of boosting iterations is also fixed to 50, which means that the boosted tree contains $50 \times 14$ trees, where 14 is the number of supported classes in the reference set. The following hyperparameters for regularisation are searched in the first step:

- max_depth $\in \{8,10\}$. The parameter defines the maximum tree depth. Deeper trees are more complex, therefore shorter trees are preferred because they reduce the probability of overfitting to the data.
- min_child_weight $\in \{5,10,30,50,60,80,100\}$ is defined to limit the tree depth. It is a threshold for the number of samples required to form a leaf node. A smaller min_child_weight value allows the algorithm to create children that correspond to fewer samples, making it more likely that more complex trees are created, which might overfit to the data.
- feature_fraction $\in \{0.6,0.7\}$ defines the fraction of features that are randomly selected to train each tree.

In the second step the learning rate and the number of trees are searched using the best hyperparameters from the first step:

- learning_rate $\in \{0.03,0.045,0.06,0.075,0.85,0.95,0.105,0.12\}$. The learning rate (sometimes called shrinkage factor) is a scalar that defines how fast the error is corrected from one tree to the next. The learning rate $v$ is inversely related to the number of trees $t$: When the learning rate $v$ is reduced to $\frac{v}{n}$, a number of $n \cdot t$ trees is required to maintain the same performance or capacity. However, high capacity makes overfitting more likely.
- n_estimators $\in \{100,150,200\}$. The parameter defines the number of boosted trees per class. The number of trees in GBDT is very critical with respect to overfitting: Adding too many trees leads probably to overfitting, it is therefore important to stop adding trees at a certain point.

For the rest of the parameters the default values are used[1].

[1] Lightgbm 3.0.0.99 documentation, `https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMModel.html` Accessed: 2020-10-07

#### 7.4.1.1 Results

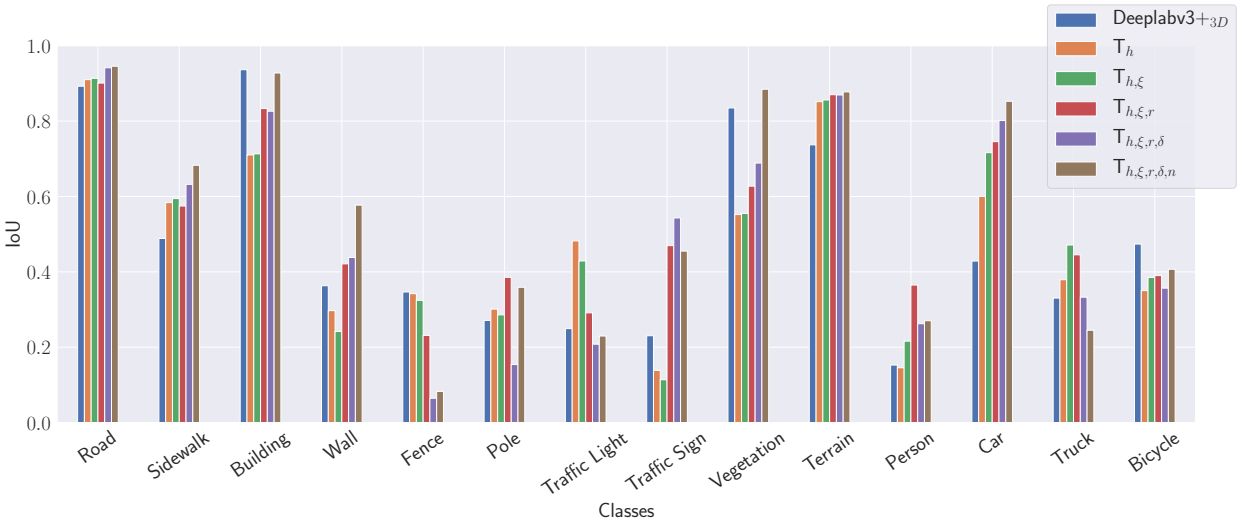



*Figure 7.6: Detailed results for the different variants of the gradient boosted decision trees in comparison to the baseline Deeplabv3+$_{3D}$*

The GBDT $T$ was trained for several trials, each using a different combination of features. Where $k \subseteq \{h,\xi,r,\delta,\vec{n}\}$ indicates the used features introduced in the Section 4.2.1.

*Table 7.2: Results on the test set for all trials using different feature combinations.*

| Method | $T_h$ | $T_{h,\xi}$ | $T_{h,\xi,r}$ | $T_{h,\xi,r,\delta}$ | $T_{h,\xi,r,\delta,n}$ |
|---|---|---|---|---|---|
| mIoU (Test set) | 0.475 | 0.487 | 0.539 | 0.509 | 0.557 |

Table 7.2 gives the mIoU for all different variants on the test set. The first tree $T_h$ was trained using the histogram only. The resulting mIoU of 0.475 might be misleading because it is very close to the original label noise of 0.481. Taking a closer look on the plot in Figure 7.6 the classifier indeed improved the IoU for several classes, e.g. *road*, *sidewalk*, *pole*, *traffic light*, *terrain*, *car* and *truck*. The second tree (green), which includes $\xi$ as a feature increased the IoU for all dynamic classes (*person*, *car*, *truck* and *bicycle*), which makes sense because the classifier is now able to recognize whether a histogram belongs to a point of a dynamic or a static object. A significant increase happens after adding the reflectance feature (red), especially for *traffic sign*, which is reasonable because traffic signs are coated with a retroreflective surface and therefore have a very high reflectance value.

The range feature (violet bars) probably leads to overfitting or mislead the classifier and decreases the performance. The reason for this might be that the measured distance is not sufficient to identify a single point class and is very scene dependent. The distance to a single point does not give any indication which class is present. However, some classes might be more likely within the same distance from the MMS in different scenes, for example, *road* and *sidewalk* points which are always close to the MMS. For these classes the IoU for $T_{h,\xi,r,\delta}$ actually increased. However, the range can be very useful in a scanstrip that provides access to successive range measurements. For example, poles can be detected very well by a sudden jump in distance that might otherwise not be detected.

Finally, after adding the normal vector (brown), the decision tree achieved the highest mIoU. The normal vector is the only feature that contains information about surrounding areas, since

it is estimated using surrounding points. The normal vector can be very useful to detect certain classes like *building*, *road* or *vegetation*. This is because it almost always points to the z-direction (upwards) on a road and is very often perpendicular to the road on a facade. In *vegetation*, the normal vectors point in all directions and are more irregular, which may also a helpful feature.

Using a GBDT with all features, the mIoU increased from 0.481 to 0.557 with pointwise correction, which leaves room for improvement. In the next sections, deep learning based approaches are presented, which compete with this learning based baseline.

### 7.4.2 Supervised Scanstrip-Based Correction

Unlike point-wise 3D label noise correction, the method evaluated in this section takes the 3D point context into account which may contain useful features. For this purpose the 3D points are represented in scanstrips. This representation allows classical 2D image processing methods. The aim is therefore to train a 2D image classifier on scanstrips containing noisy labels to predict the (correct) labels. In this section the classifier Scanstrip Network (SNet) which was introduced in Section 4.2.1 is used.

#### 7.4.2.1 Training Parameters

The hyperparameters of SNet are the input size $s$ and depth $b$. Where $s$ defines the windows width and height and $b$ the number of channels which are used for the input $x$. The training is done in two steps: In the first step, the size $s$ is fixed to 64 (pixels) and the Scanstrip Network (SNet) is trained for several trials using different input features. In the second step the configuration with the highest performance on the **validation set** is used in order to tune $s$.

All features were introduced in detail in Chapter 4.2.1. The training is first done using only the histogram information $h$. As there are 19 different classes the input size is $[s \times s \times 19]$ for $\text{SNet}_h$, where the index in SNet indicates the used features. In the following trials, the campaign count $\xi$, the reflectance $r$, the range $\delta$ and the normal vector $\vec{n}$ are added one after another in order to judge the different performances. Consequently the network $\text{SNet}_{h,\xi,r,\delta,\vec{n}}$ has an input size of $[s \times s \times 25]$.

In the second step the network is trained and evaluated on different input sizes $s \in \{32{,}128{,}256\}$. In all trials, a fixed batch size of 32 is used. Each minibatch sample is gathered by randomly cropping a window from one of the scanstrips in the training set. Obviously, windows that do not contain label information are discarded. In every trial the training is done for 40k iterations. Early stopping is achieved by storing the weights achieving the highest performance on the validation set.

#### 7.4.2.2 Validation set

The training and test set split from Chapter 7.3 is used, which makes it possible to compare this method with the point-wise label correction. However, since the training of a DCNN is computationally very expensive, this method is not trained using cross-validation. Instead, the training set is divided once into a training set and a validation set. To ensure that the validation set in the ablation study is always the same regardless of the input size, the training dataset is cropped into windows of size $[n \times 256 \times 256 \times b]$, which is the maximum window size that is used in step two of the training. If a window at the edge of the scanstrip is smaller, it is padded by zeros. The training and validation sets are then created by splitting the windows into two subsets such that the validation set supports each class by 10% - 20%. This results in a split of 58% for the training

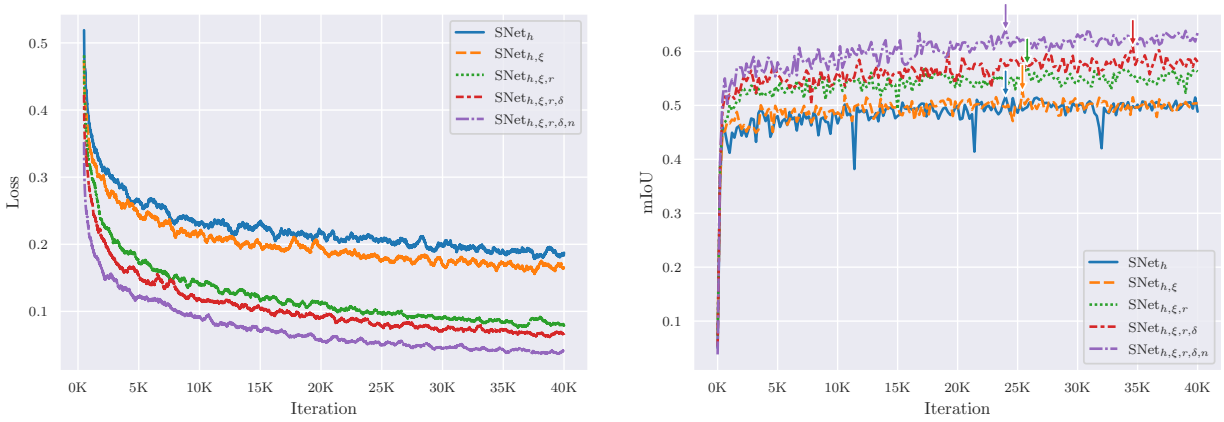



*Figure 7.7: Loss on training set (left) and mIoU on the validation set (right). The arrows in the right plot indicate the global maximum. All plots were smoothed with moving average for better visibility.*

set, 16% for the validation set, and 26% for the test set. When a model is trained with a smaller window size $s \in \{32,64,128,256\}$, the validation set is simply cropped to the corresponding size.

#### 7.4.2.3 Results

The diagrams in Figure 7.7 show that the use of all features leads to the highest mIoU on the validation sets. This is not an obvious result, because additional features may lead to a higher degree of overfitting, especially if the training set is very small. The left diagram in Figure 7.7 shows that each time a feature is added, the loss on the training set decreases more rapidly. Interestingly, the largest decrease happens after adding the reflectivity feature (green curve). This was also observed in the point-wise correction in the previous section. A possible reason for this is that the histograms and campaign count may not describe the objects and their context well enough.

As can be seen from the diagram on the right, the performance on the training set can be transferred to the generalization capability of the networks on the validation set. The results on the test set are shown in Table 7.3. The observations from the diagrams can also be transferred to the test set. Interestingly, adding the range feature did not decrease the performance as it did in the point-wise correction. This may be due to the fact that the range makes it easier to identify poles and trees due to sudden changes in the measured distance. Table 7.3 contains an additional test (see SNet$_{\xi,r,\delta}$) with all features except the histogram. It shows the worst performance on the test set and makes it clear that the transferred labels contain very valuable information.

*Table 7.3: Ablation study for the correction of label noise with scanstrip network (SNet). The indices indicate the used features.*

| Method | SNet$_h$ | SNet$_{h,\xi}$ | SNet$_{h,\xi,r}$ | SNet$_{h,\xi,r,\delta}$ | SNet$_{\xi,r,\delta}$ | SNet$_{h,\xi,r,\delta,n}$ |
|---|---|---|---|---|---|---|
| Training mIoU | 0.669 | 0.711 | 0.798 | 0.850 | 0.811 | **0.918** |
| Validation mIoU | 0.516 | 0.526 | 0.575 | 0.609 | 0.555 | **0.654** |
| Testing mIoU | 0.528 | 0.562 | 0.619 | 0.631 | 0.51 | **0.659** |

As the network SNet$_{h,\xi,r,\delta,n}$ scored best on the validation set, it is selected to evaluate different input window sizes. Note that the test mIoU has not been considered for selecting the model to

*Table 7.4: Ablation study for finding a good input window size for network $SNet_{h,\xi,r,\delta,n}$. For better readability the indices are omitted.*

| Method | $SNet^{(32)}$ | $SNet^{(64)}$ | $SNet^{(128)}$ | $SNet^{(256)}$ |
|---|---|---|---|---|
| Training mIoU | 0.871 | 0.891 | **0.955** | 0.907 |
| Validation mIoU | 0.633 | 0.654 | **0.665** | 0.641 |
| Testing mIoU | 0.665 | 0.659 | **0.667** | **0.667** |

avoid overfitting. The results for $SNet_{h,\xi,r,\delta,n}$ are shown in Table 7.4. For better readability the network that uses all features will be named SNet without indices from now on unless otherwise stated. The used input window size $s$ is indicated by $SNet^{(s)}$. The symbol of $SNet^{(32)}$ reads therefore as Scanstrip Network (SNet) with an input size of $[32 \times 32 \times 25]$.

Overall, the networks performed very similarly for different window sizes. Large differences can be seen between training mIoU and validation mIoU. A high training mIoU as opposed to a low validation or testing mIoU indicates overfitting. The best performing model is $SNet^{(128)}$ with a very small margin. As it is undecided which model performs best, both $SNet^{(128)}$ and $SNet^{(256)}$ are used in the following experiments. However, it should be emphasized that the mIoU is significantly higher than the point-wise correction using GBDT. The pointwise correction reached a value of 0.557 and the scanstrip-based one reached a value of 0.667. This means that the proposed method was able to predict the classes in 3D at a similar level as estimated in 2D before label transfer (0.661).

#### 7.4.2.4 Comparison

To show how well SNet performs compared to other architectures, it is compared to four different semantic segmentation networks. The first is HRNet (Sun et al., 2019) and the second one is Deeplabv3+ (Chen et al., 2018). Deeplabv3+ was trained using the Xception network as feature extractor (Chollet, 2017). Finally, SNet is compared to FCN (Long et al., 2015), which is one of the oldest models for semantic segmentation, and the U-Net (Ronneberger et al., 2015), which is very similar to SNet. Note that originally U-Net does not use padding in the convolution, resulting in a prediction that is smaller than the input. Therefore, the network has been slightly modified by using zero padding in the convolutional layers to increase the output size to the original input size.

All networks are trained with the same hyperparameters as SNet. Since it is undecided which window size $s \in \{128,256\}$ is the best, they are trained in two variants. One with a randomly cropped window of size $[128 \times 128]$ and the other with a size of $[256 \times 256]$. All networks are trained using the same training, validation and testing data as SNet. All networks are also optimized using Adam, with the same learning rate of 0.001 for 40k iterations with a batch size of 32 and the cross entropy as loss function (see Equation 2.16). The training is augmented by random horizontal or vertical mirroring of the input. As the models are larger than SNet with respect to the numbers of trainable parameters, two strategies for early stopping were used to mitigate overfitting: The first strategy, which was also used for SNet, is to train each network during the entire 40k iterations. The weights that achieved the highest validation mIoU are used for testing. The second strategy stops the training process if the network does not increase the mIoU on the validation set for 4000 iterations. This should make it more likely that weights from an early stage of the training process are used. The strategy that achieves the best results on the test set is shown in the results, with

*Table 7.5: The performance of SNet compared to other network architectures. Results marked by an asterisk indicate that the best performance on the test set was achieved due to early stopping with the second strategy*

| Resolution 256 × 256 | | | | | |
|---|---|---|---|---|---|
| Method | SNet | HRNet | FCN | U-Net | Deeplabv3+ |
| Training mIoU | 0.907 | 0.886 | 0.631 | 0.780 | 0.852 |
| Validation mIoU | 0.633 | 0.624 | 0.631 | 0.579 | 0.617 |
| Testing mIoU | **0.665** | 0.628 | 0.549 | 0.630 | 0.620 |
| Resolution 128 × 128 | | | | | |
| Training mIoU | 0.955 | 0.892* | 0.902 | 0.797 | 0.850* |
| Validation mIoU | 0.655 | 0.624* | 0.588 | 0.591 | 0.589* |
| Testing mIoU | **0.667** | 0.612* | 0.606 | 0.614 | 0.583* |
| parameters | 4M | 9.5M | 34M | 31M | 41M |

an "*" indicating the second strategy. The combined results for both windows sizes are shown in Table 7.5.

Table 7.5 shows that SNet achieved the highest performance among all networks, regardless of the window size. There could be several reasons for the poor performance of the other networks on the scanstrip dataset. The first is that scanstrips have a very low resolution, which means that some classes like *pole* are only 1-2 pixels wide. Deeplabv3+ is designed to segment larger areas than that, which can be too inaccurate for the scanstrips. The low resolution feature map in the last layers of Deeplabv3+ is only interpolated to create the predictions. Networks such as U-Net or SNet are better suited for such a task, because they use transposed convolutions with skip connections, which can contribute to a higher pixel accuracy. The last reason might be that the other networks have too many trainable parameters. The last line in Table 7.5 shows that SNet has the smallest number of trainable parameters, which is good if the network is trained on a small dataset as it prevents overfitting. For example, SNet has almost the same depth in terms of hidden layers as U-Net, but 7.75 times fewer parameters. Overall, all compared networks performed better using the larger window size. The test shows that the proposed network architecture is well suited for this task.

### 7.4.3 Semi-Supervised Scanstrip-Based Correction

In the previous sections it was shown that it is possible to correct labelling errors by supervised learning. This section shows how even better performance can be achieved when the scanstrips are corrected using Semi-Supervised Learning (SSL).

#### 7.4.3.1 Training and Results

The training strategy for $\text{SNet}_{SSL}$ network was already explained in detail in Section 4.2.2. The **first step** is very similar to the supervised methods. As the entire dataset is used here and is sufficiently large, the validation set is selected by simply removing 8 randomly selected scanstrips from the training set, which have sufficient support for each class. Otherwise, no test set is needed in this step, because the weights with the best performance are selected only based on the highest

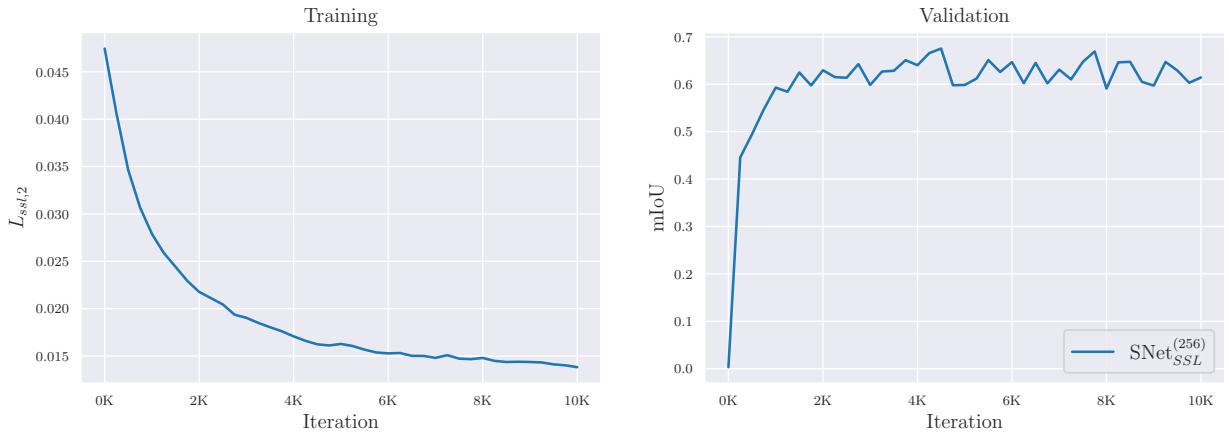



*Figure 7.8: The Loss of* SNet$^{256}_{SSL}$ *in the second training phase (left) and mIoU on the validation set (right).*

mIoU of the validation set. The training is performed for 100k iterations with a batch size of 32 and a learning rate of 0.001 using the Adam optimizer. The selected size of the input windows is $[256 \times 256 \times 6]$, which are randomly cut out of a randomly selected scanstrips. The data is augmented by randomly mirroring along the vertical or horizontal axis. In addition, a 25% drop-out is used after each transposed convolution.

In the **second step** the weights which achieved the highest mIoU on the validation set in the first step are used. Apart from that, the network is altered as described in Section 4.2.2 in order to retrain it on the human annotated reference set. The training is done for 10k iterations with the same training, validation and testing split used in in the Section 7.3 before. The training is visualized in Figure 7.8. Comparing Fig. 7.8 to Fig. 7.7, one sees that the loss of the SSL version is about an order of magnitude lower than that of the supervised version. The validation plot in Fig. 7.8 shows that the network reaches a validation mIoU of **0.686** after 4500 iterations. These weights are used for testing. The results in Figure 7.9 show that the semi-supervised network SNet$^{(256)}_{SSL}$ performs better for almost every class than the supervised version SNet$^{256}$. It is able to achieve a mIoU of **0.709** on the test set, compared to 0.667 for the supervised version. It is particularly noticeable here that the SSL variant performance much better for the *fence* class and even outperforms *Deeplablv*3+$_{3D}$, which is predicted very poorly by the methods tested so far. The reason for this could be that SNet$^{(256)}$ overfits to a particular *fence* type in the training and validation sets that does not appear in the test set. This shows that SNet$^{(256)}_{SSL}$ is capable of extracting new information from the first training step and generalizes significantly better than the supervised version after the second training step.

### 7.4.4 Qualitative Evaluation

Three models for correcting labeling errors were presented, which were either based on point-wise predictions or scanstrip based. For the latter, it was shown how a semi-supervised strategy can be implemented using the entire "noisy" dataset. In this section, all results are qualitatively evaluated by comparing them on the same test data.

The images in 7.10 show three different scenes, one for each column. The first row shows the input point cloud for each scene, colored by the height of the point cloud. The following rows show the result for each method presented in Chapter 4 for each point cloud. The first scene shows a narrow street surrounded by buildings. On the right side there are some parked cars. The scene contains only static objects. The second scene shows an open area with an intersection, controlled

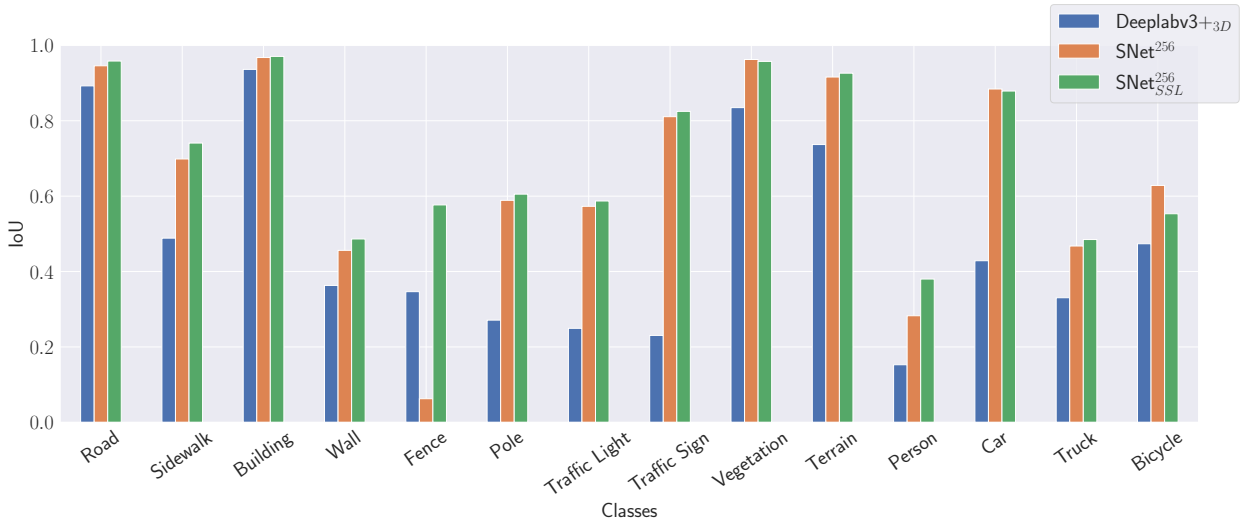



*Figure 7.9: A comparison of the IoU on the test set for every class between the naive label transfer (blue), supervised correction (orange) and semi-supervised correction (green)*

by traffic lights, and buildings in the background. There are also streets, trees and cars in the background. In the foreground, a person is walking on the sidewalk. This scene contains more dynamic objects because the cars are not parked in a parking lot and the pedestrian is moving. The third scene shows a building with a sidewalk and some parked cars and trees in front of it. This scene is intended to show the effects of errors related to the labelling policy, as the trees occlude the building facades.

**The first column** shows a static scene, which is well suited for the baseline method, because the label transfer does not suffer from dynamic occlusions. Looking at the naive result showing the baseline, most objects are recognized. However, the traffic sign in the left part of the image remains undetected and is labelled as a *building*. Cars and bicycles also suffer from label bleeding. The gradient boosted decision tree was able to restore the traffic sign in the left part of the image and remove the label bleeding of the cars. However, it introduced several new errors, such as *sidewalk* labels in front of the cars and many misclassifications on the facades, poles and traffic signs. The supervised scanstrip net ($\text{SNet}^{(256)}$) looks a bit better, it makes fewer misclassifications than the GBDT. However, *road* is very often misclassified as a *sidewalk*. The semi-supervised trained scanstrip network ($\text{SNet}^{(256)}_{SSL}$) achieved visually the best results. The cars do not suffer from bleeding labels, and the sidewalks looks cleaner. However, the network could not identify the terrain below the tree in the center of the image, and some objects in the middle left part of the image are "speckled" with *sidewalk* labels.

**The second column** shows a more dynamic scene. As expected, the baseline fails completely in these cases due to dynamic occlusions. For example, the person in the front is not detected at all. The intersection is covered with *car* labels and some waiting cars in the background are labelled as *building*, *road* and *sidewalk*. Also the sidewalk in the foreground is not detected, and many parts of the scene in the background are not classified due to regular occlusion. Overall, the result is really bad. The GBDT shows that the classifier can correct most of the errors, but as in the first scene, it suffers from misclassifications for almost all objects. Interestingly, the tops of many cars are classified as *person* in the background. A possible reason for this is that in the 2D images a person crossed the street, which accumulated *person* votes in these histograms. The supervised SNet in the fourth row suffers from similar errors, but the overall result looks much more consistent. However, the network introduces a certain degree of label bleeding, i.e. the building is covered with

*person* labels and the trees on the right side are labelled as *traffic sign* (orange). Finally, the SSL network provides the cleanest results. The building facade is free of label bleeding and most cars are correctly detected. However, $\text{SNet}_{SSL}^{(256)}$ has misclassified some parts of the sidewalk and the pole.

**The third column** shows an example of a building occluded by trees. The baseline shows the typical errors of having the facade covered with *vegetation* labels. Also, many parts of the building could not be classified due to regular occlusions and are therefore removed from the result. The presented methods show better results. While the GBDT and $\text{SNet}^{(256)}$ were not able to remove all *vegetation* labels from the facade, the semi-supervised method gives a really clean result. Visually, $\text{SNet}_{SSL}^{(256)}$ is able to remove errors, all of which are likely to be policy-based, since the facades in the last row have no misclassifications.

Figures 7.11, 7.12 and 7.13 show the final results of $\text{SNet}_{SSL}^{(256)}$ for the scenes depicted Fig. .7.4 and Fig. 7.5. The image in Fig. 7.11a shows very severe case of dynamic occlusion where a road edge that is covered with grass (terrain) was labelled as *car*. The result after the label noise cleaning shows no signs of dynamic occlusion. The network removed the errors successfully from the data and even recovered the sidewalks in the background.

The image in the second example (Fig. 7.12a) shows a scene with typical calibration and labelling-policy errors. The upper edges of the building are labelled as *sky* (calibration error) and the facade is sprinkled with *vegetation* labels (label-policy error). The picture on the right shows the same facade after the cleaning process. The correction of the *sky* labels from the data is not necessarily as trivial as their removal. As can be seen, the network has assigned the correct class to the edges of the facade and completely removed the label policy errors. Overall, the facade looks clean and does not show any errors.

$\text{SNet}_{SSL}^{(256)}$ shows a similar performance when it comes to regular and self-occlusions. The scene in 7.13a is rendered from a bird's eye view. It shows a sidewalk with heavy label bleeding around cars (blue), bicycles (dark red) and pedestrians (red). Although the cars and bicycles are parked and do not suffer from dynamic occlusions, they are very difficult to detect due to the amount of label bleeding. A more detailed example is shown in Figure 7.14. It shows a number of extracted bicycles and cars from this scene. In the left image the bicycles are hard to recognize. The result of $\text{SNet}_{SSL}^{(256)}$ on the right shows that they can be recognized much better in the point cloud. This type of error is very critical for further processing, e.g. for object instantiation.

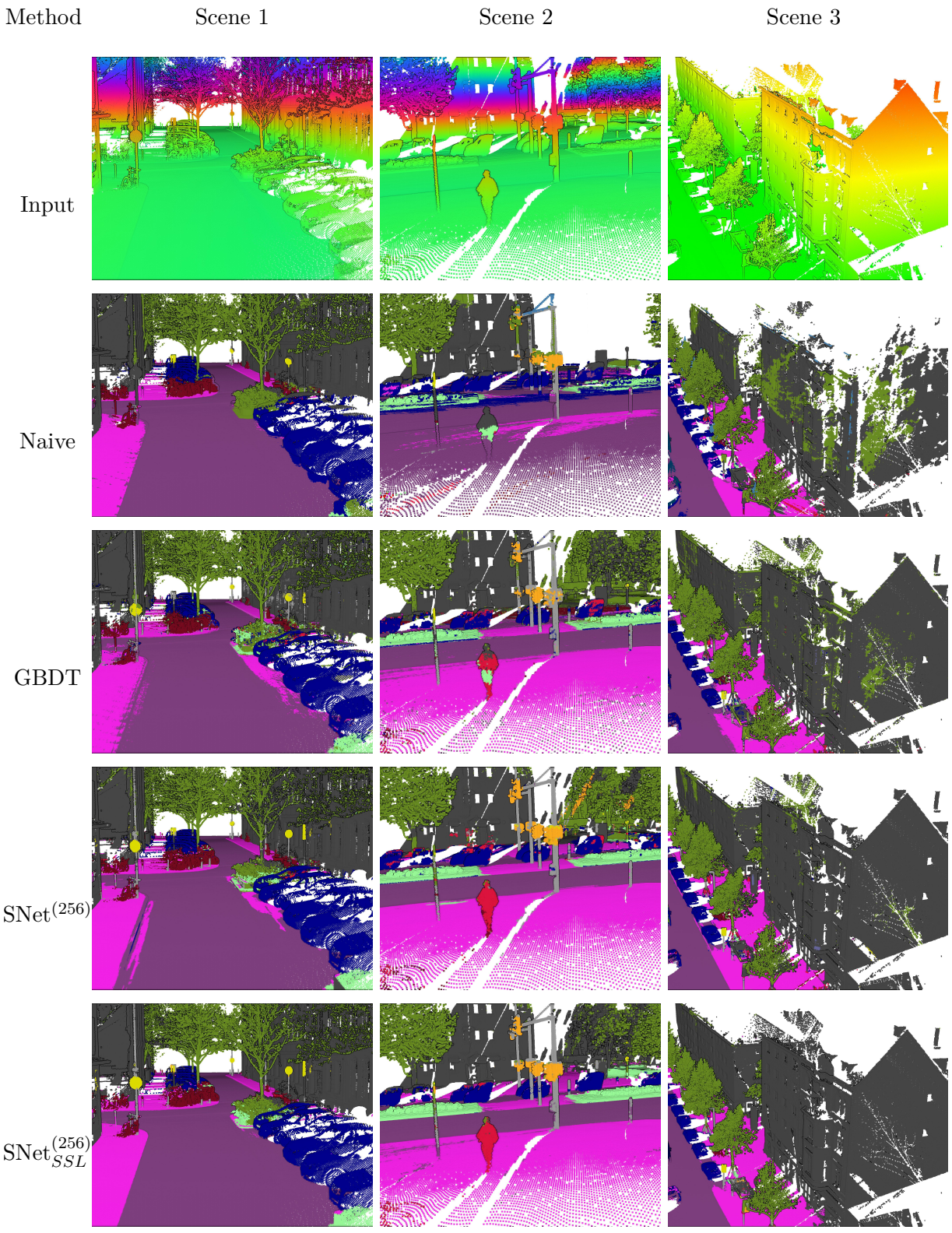
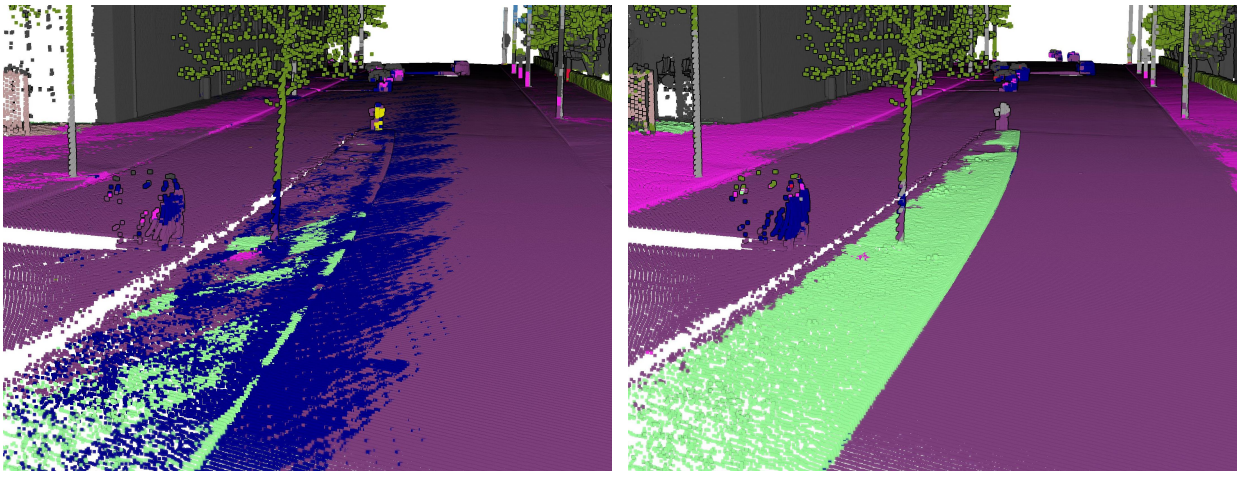
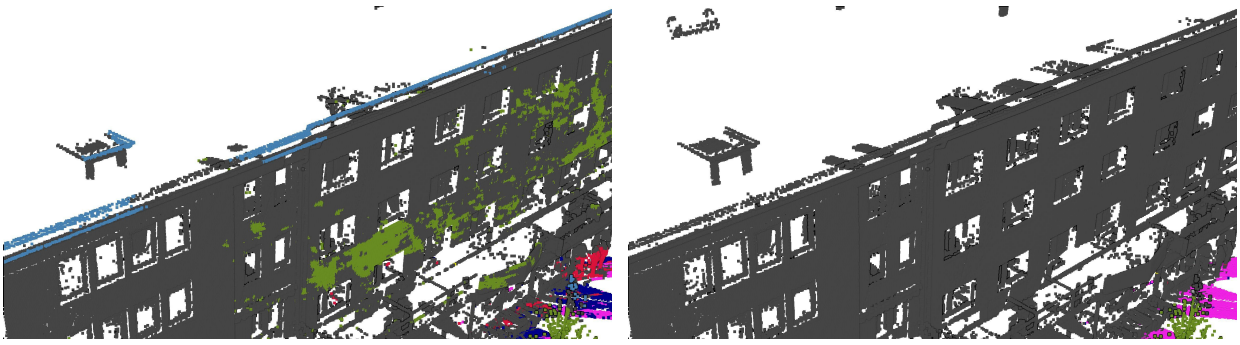



*Figure 7.10: Qualitative comparison of the presented methods for three different scenes. Each column represents a scene, which is colored differently per row according to the results of the individual methods.*



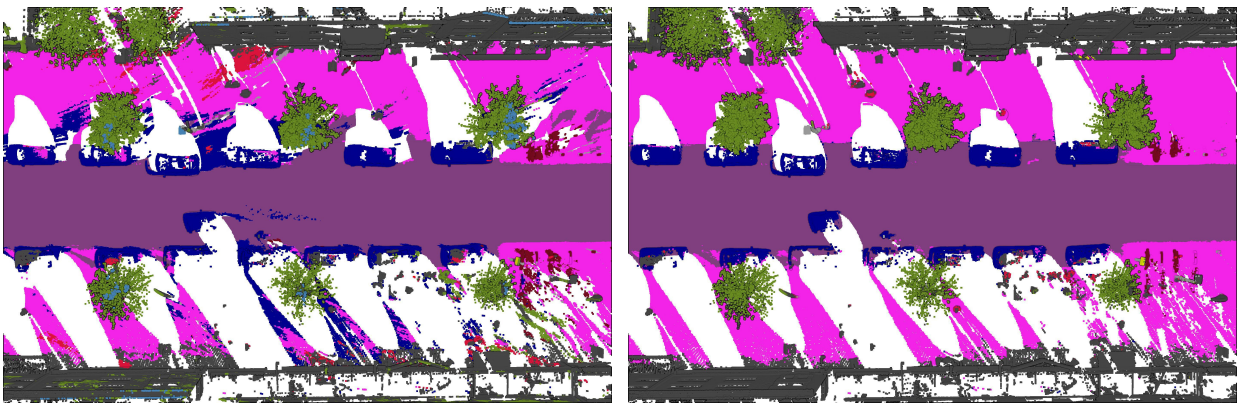
*(a)* Naive *(b)* $\text{SNet}_{SSL}^{(256)}$

*Figure 7.11: Comparison of the naive result to* $\text{SNet}_{SSL}^{(256)}$ *in a scene shown in Fig. 7.4 containing errors due to dynamic occlusions.*



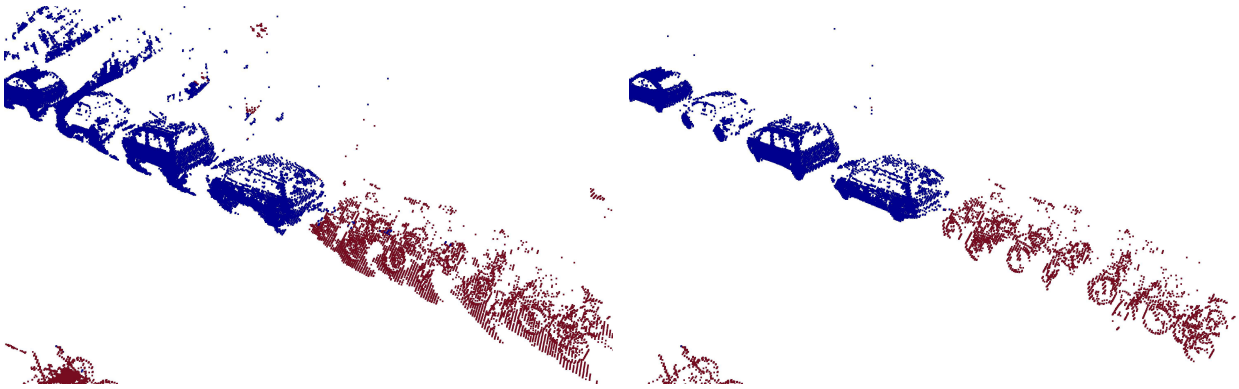
*(a)* Naive *(b)* $\text{SNet}_{SSL}^{(256)}$

*Figure 7.12: Comparison of the naive result to* $\text{SNet}_{SSL}^{(256)}$ *in a scene shown in Fig. 7.4 containing label policy and calibration errors.*



*(a)* Naive *(b)* $\text{SNet}_{SSL}^{(256)}$

*Figure 7.13: Comparison of the naive result to* $\text{SNet}_{SSL}^{(256)}$ *in a scene containing regular- and self-occlusions.*



*(a)* Naive *(b)* $\text{SNet}_{SSL}^{(256)}$

*Figure 7.14: Cars and bicycles extracted in the scene shown in Figure 7.13.*

### 7.4.5 Conclusion and Discussion

In the previous sections it was shown how to transfer labels in images classified with a pretrained DCNN into the 3D domain by exploiting the geometric correspondence between image and laser rays in a fully calibrated system. However, due to various errors, the transfer resulted in labelling errors in the 3D point cloud, which means that many 3D points are assigned to the wrong class. By using two reference sets, one in 2D and one in 3D, various sources of error could be identified by comparing the label confusion before and after the transfer. Furthermore, the amount of labelling errors was estimated by comparing the transferred labels to the human annotated reference set in 3D. This served as a baseline for the methods that were tested afterwards.

As the number of 3D reference points is very limited, an optimal way to split the annotated data into training and test sets was shown. This method splitted the dataset by searching for "representative" subsets, keeping both sets locally separated. A "representative" subset means that the labelling errors in each subset is very similar to the estimated total labelling errors.

Three methods were introduced for label error correction, all of which had to cope with a very small annotated dataset. First, a relatively simple point-by-point correction based on gradient boosted decision trees was introduced. Since this method is only point-wise, it helped to understand the information gained from the different features introduced to correct labelling errors. For example, it was shown that the measured distance itself does not contribute to the identification of a class unless the distances are considered in context with those of other nearby points. This problem is solved by the scanstrip-based representation used to train a DCNN to learn how to correct the wrong labels. The introduced network was fine-tuned and later compared to other semantic segmentation networks, which were also fine-tuned to the scanstrip dataset. It was shown that the presented SNet achieved the highest scores on the test set compared to all other methods.

Finally, a strategy for semi-supervised semantic segmentation was introduced. This method achieved significantly better results than all other networks by learning how to semantically segment scanstrips from the entire (noisy) dataset. The reference dataset was only used for fine-tuning. To illustrate the size of the training set, these 51M annotated points correspond to about 24 finely annotated images from Cityscapes. While Cityscapes contains 25k annotated images that are necessary to train a DCNN.

## 7.5 Multi-View Error Correction

The methods described in the previous sections have dealt with different types of label noise with reasonable success. However, as can be seen from Table 7.1, all of these methods have only indirectly dealt with the cause of label noise after the labels have already been aggregated in 3D. In this section, Multi-View Networks (MVNets) are used to directly treat the cause of of wrong classifications in 2D images. The advantages are manifold: first, it is shown that pixels misclassified in the MMS images can be corrected. Second, a pretrained DCNN can be fine-tuned on the corrected data, reducing the domain gap in the 2D MMS images. Finally, in the next section, it is shown that this method extends the existing version of label transfer, where classified pixels are simply mapped from 2D to 3D points by naive majority voting. Instead, the network can learn to detect errors in the 2D predictions and correct them during the transfer, resulting in less misclassification in the 3D data.

The structure of this section is similar to the previous one. First, a baseline is presented along with the training and a test set. For training, validation and testing only the 2D reference images shown in Figure 6.3 are used and no 3D labels. After the introduction of the baseline this section follows the steps defined in Section 4.3.1.2:

1. First, a set of initial predictions are generated with a pretrained DCNN for the MMS dataset.

2. Then, the MVNet is trained using only sparse labels.

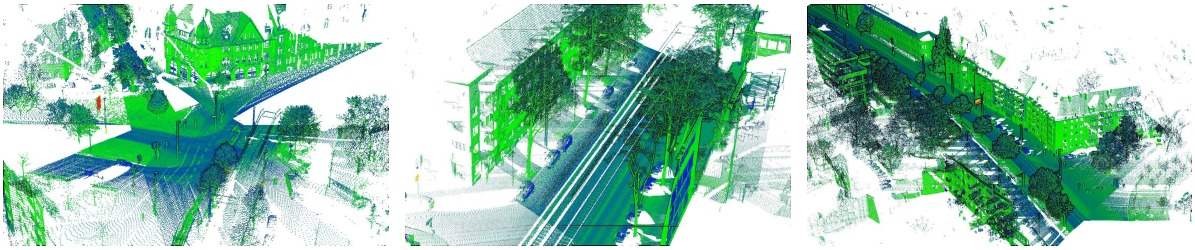
3. The trained MVNet is used to correct all initial predictions.

4. Finally, the pretrained DCNN is fine-tuned using the corrected predictions.

To show the superiority of the Multi-View Network approach, an ablation study is conducted comparing Multi-View Networks to Single-View Networks (SVNets). Here, an SVNet can be defined, as any network that has only access to one image at a time. All steps are concluded with quantitative and, where available, qualitative results. Finally, the conclusion of this section shows a comparison of all methods evaluated up to that point.

### 7.5.1 Baseline

In Section 7.2 it was shown that Deeplabv3+ suffered from a moderate domain gap when the pretrained network was applied on the MMS images. The estimated mIoU on the MMS images was about 0.68, which is worse than on the original Cityscapes test set with an mIoU of 0.81. However, Deeplabv3+ is available in different variants, which means that the backbone (encoder part) is interchangeable. To obtain the results in Section 7.2, Deeplabv3+ was used with an xception-71 backbone, which is relatively large and consumes a lot of GPU memory. Since the networks need to be re-trained in the following procedure, the xception-71 backbone is too large for a single Nvidia Titan X GPU to fine-tune it, even on rescaled MMS images. To make the results as comparable as possible, Deeplabv3+ with an xception-65 backbone is used here, meaning it has only 65 xception layers instead of 71. The weights[2] are available in the tensorflow repository. On the MMS image dataset, it achieves a mIoU of 0.62 compared to 0.68 with xception-71. The following results are compared to the mIoU of 0.62 obtained with Deeplabv3+$_{2D}$ which used xception-65 as backbone.



*(a)* Scene 1 *(b)* Scene 2 *(c)* Scene 3

*Figure 7.15: Three point clouds show the scenes in which the MMS images were acquired for training and testing. The images from scene 1 are used for training. The images from scene 3 are used for validation and testing. All point clouds are colored according to reflectance values.*

### 7.5.2 Training, Validation and Test Sets

To meet the requirements for training and test sets, as introduced in Section 7.4.2.2, the training and test sets are spatially divided, i.e., the data for the training set is collected at a different location than the on of the test set. The 23 human annotated images are collected at three different locations belonging to at least three different scanstrips, see Fig. 7.15. Scenes one and two are used for training. The training set contains all static classes. Scene one is a wide and open area containing mainly *building*, *road*, *car*, *pole*, *traffic sign* and *traffic light* classes. Scene two shows a narrow urban street. In this scene, all other classes, such as *wall*, *fence*, *bicycle* and *person*, are available. Scene three is used for validation and testing (note this scene also intersects with the test set from Section 7.4.2.2). It is also shown in Figure 7.10 (column two) and Figure 7.11 and contains all classes, consisting an open area and a narrow road.

In the first phase of this method, a MVNet must be trained to correct erroneous predictions of the pretrained Deeplabv3+$_{2D}$ due to domain shifts. The MVNet is trained, validated, and tested only on image pixels associated with a 3D point from the point clouds in Figure 7.15. In the second phase, the DCNN is fine-tuned with the corrected data. As all 23 labelled images are used to train and test the MVNet, fine-tuning the DCNN would require additional labelled data for validation and testing. As not every 2D pixel is associated to a 3D point, there are remaining (leftover) pixels that can be used to validate and test the fine-tuning of Deeplabv3+. To illustrate the different training, validation and test sets and all leftover labels, the Table 7.6 shows the support for each class for each subset

Figure 7.16 summarises and visualizes the process of creating all sets. It shows that the entire reference set containing all 23 images is first divided into the training and test sets based on their location. To create a validation set for training the Multi-View Networks, a portion of 50% of the samples of each class was randomly selected from the testing set. All points associated with a 3D point ("Associated" boxes) are used for training, validation and testing the Multi-View Network. The remaining ("leftover") annotations are later used for the fine-tuning step of the DCNN. In this way, no label is used more than once and there is no overlap between the different subsets.

Because of dynamic occlusions, only classes belonging to static objects are used for training and testing the Multi-View Networks. Table 7.6 shows that about 16.16% of the labels are used for training and testing. This is roughly equivalent to only 3-4 labelled ground truth images. The training and test sets contain a total amount of 12.4 M annotated examples. The reason for the relatively small size of the training and test sets is the sparseness of the point cloud. Additionally,

[2] `http://download.tensorflow.org/models/deeplabv3_cityscapes_train_2018_02_06.tar.gz`

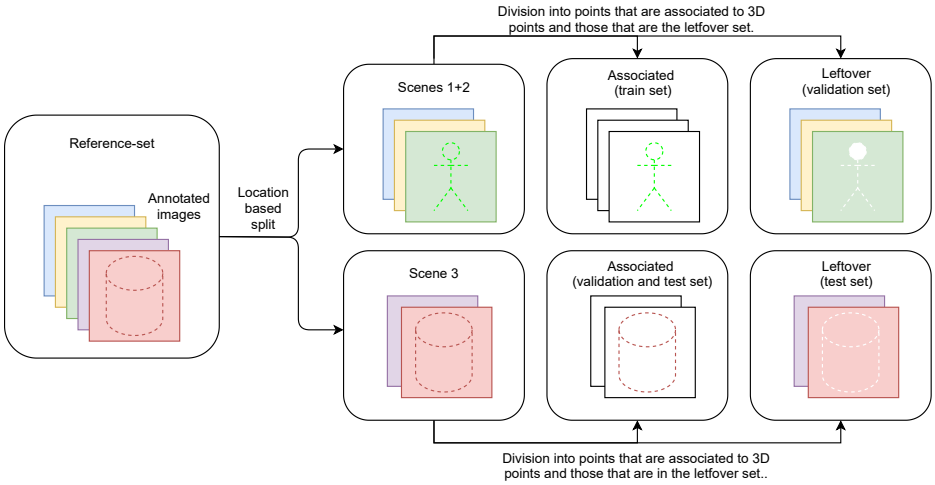



*Figure 7.16: Schematic overview visualizing the creation of the train, validation and test set for the Multi-View Network (Associated). The remaining annotations are later used in the fine-tuning step of the DCNN (leftover).*

| Name | Road | Sidewalk | Building | Wall | Fence | Pole | T.Light | T.Sign | Veg. | Terrain | Used |
|---|---|---|---|---|---|---|---|---|---|---|---|
| GT Set [k] | 33261 | 8650 | 16959 | 324 | 293 | 1344 | 323 | 81 | 13840 | 1460 | 76536 |
| Train Set [k] | 3405 | 1179 | 3632 | 82 | 29 | 244 | 85 | 6 | 823 | 150 | 9635 |
| Train Set[%] | 10.24 | 13.64 | 21.42 | 25.2 | 10.01 | 18.15 | 26.21 | 7.95 | 5.95 | 10.25 | 12.59 |
| Test Set[k] | 1331 | 381 | 344 | 4 | 10 | 26 | 10 | 4 | 546 | 74 | 2730 |
| Test Set[%] | 4.0 | 4.4 | 2.03 | 1.3 | 3.33 | 1.91 | 3.19 | 5.14 | 3.95 | 5.08 | 3.57 |
| Leftover [k] | 28525 | 7090 | 12983 | 238 | 254 | 1075 | 228 | 70 | 12471 | 1236 | 64170 |
| Leftover [%] | 85.76 | 81.96 | 76.55 | 73.5 | 86.66 | 79.93 | 70.6 | 86.91 | 90.11 | 84.67 | 83.84 |

*Table 7.6: The table shows the support for the training and test subsets. The first row shows the total amount of labelled pixels for each class. The following rows show the support for each subset in thousands (k) and percent of the total. The last set (leftover) shows the amount of data that is not used for training or testing because it is not associated with any 3D point.*

some regions of the ground truth images are not covered at all. Also, a 3D point was only mapped into a 2D image if it is **within 60 m** of the projection centre. Otherwise, calibration errors may become too dominant. However the benefit of doing this is that the leftover set still contains 83.84% of the ground truth labels, which can be used for the evaluation of the fine-tuned DCNN model after the correction process. As an additional note, 17 of the total 23 ground truth images are part of the training set and 5 images are part of the test set.

The last step is to fine-tune Deeplab on the corrected data. For this the leftover set is split into a validation and a test set. Here, leftover labels from scenes 1 and 2 are used for validation and the leftover from Scene 3 are used for testing the final model, see Fig. 7.16. In this way the DCNN is validated on pixels that were never used in the multi-view training and testing steps. Additionally it is made sure that the DCNN does not overfit because the test set contains images from a different location as the validation set.

### 7.5.3 Training Procedure

This section describes the training procedure for the Multi-View Networks. The correction network $\text{MVNet}_{\text{DL}}(z_i) = \widetilde{y_i}$ is trained to correct the predictions of Deeplabv3+ on the MMS image dataset. Here $z_j$, as described in Section 4.3.1.1 contains the multi-view RGB image patches for the corresponding 3D point $x_i$ and the lists of image features $\hat{d}_j$ and $g_j$. Where $\hat{d}_j$ are the softmaxed scores made by the pretrained Deeplabv3+ and $g_i$ is a list containing the distances between the 3D point and the projection center corresponding to each image. The network $\text{MVNet}_{\text{DL}}(z_i) = \widetilde{y_i}$ must learn to map the observations $z_i$ to a list of ground truth labels $y_i$. Here, $y_i$ is a list containing a ground truth label for each multi-view observation, and $\widetilde{y_i}$ is a list containing the corresponding predictions. Since $y_i$ is sparse due to the lack of ground truth labels in all associated multi-view images, the missing values are filled with placeholders so that $y_i$ and $\widetilde{y_i}$ both have the same length.

The network is trained for 100,000 iterations and evaluated against the validation set. A small dropout rate of 10% is chosen for training. Empirically, it could be observed that the random deletion of list entries in $z_i$ during training is very helpful to augment the data. Therefore, a random number of list entries were deleted at each training step as long as at least two entries were still present. The network is optimised with Adam optimizer using default parameters with a learning rate of $\lambda = 0.001$ and $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The window and mini-batch size $w$ and $b$ are tuned by grid search with $w \in \{0,2,4,32\}$ and $b \in \{8,16,32,64\}$, where $w = 0$ means that the networks do not have access to any RGB information. In all scenarios, the evaluation step is performed every 1000 iterations on the entire validation set. Loss functions that have proven useful in this scenario are the Generalised Dice Loss (GDL) by Sudre et al. (2017) and the Tversky loss (T) by Salehi et al. (2017), as they are made for highly imbalanced data with multiple classes.

The Dice score is a metric which is defined as follows:

$$\text{DICE} = \frac{2TP}{(TP + FP) + (TP + FN)} \tag{7.4}$$

Since this metric is not differentiable the GDL is used instead:

$$\text{GDL}_i = 1 - 2\frac{\sum_c \sum_n \widetilde{y}(c,n)_i y(c,n)_i}{\sum_c \sum_n \widetilde{y}(c,n)_i + y(c,n)_i}, \tag{7.5}$$

where $\widetilde{y}(c,n)_i$ is the predicted probability for the $c$-th class at the $n$-th entry of the the list $\widetilde{y}_i$. The ground truth is given by $y$ as a one-hot encoded vector of length $c$ with a one for the reference class at the entry $n$. The sums are taken over all classes $c \in C$ and all list entries $n \in N$. The Tversky loss $T(\alpha,\beta)$, on the other hand, sets different weights $(\alpha,\beta)$ for false negatives (FN) and false positives (FP).

$$\text{T}(\alpha,\beta) = \frac{TP}{TP + \alpha FN + \beta FP}, \tag{7.6}$$

where each term can be calculated analogously to Equation 7.5 as follows:

$$TP = \sum_c \sum_n \widetilde{y}(c,n)_i y(c,n)_i \tag{7.7}$$

$$FN = \sum_c \sum_n \widetilde{y}(c,n)_i (-y(c,n)_i + 1) \tag{7.8}$$

$$FP = \sum_c \sum_n (-\widetilde{y}(c,n)_i + 1) y(c,n)_i \tag{7.9}$$

A very important part of the Multi-View Network is the self-attention mechanism, which allows the network to relate corresponding multi-view observations to each other, so that the information

from one observation can be propagated to all others, helping the network to correct the predictions. Recall that the network $\text{MVNet}_{\text{DL}}(z_i)$ receives a list of multi-view observations and makes a prediction for each list item in $\widetilde{y}_i$. Without the self-attention mechanism, the network will treat each list element independently of the other elements. Thus, if the self-attention mechanism is removed, the network collapses into a Single-View Network (SVNet) which has exactly the same number of layers and trainable parameters as the original network, except for the trainable self-attention parameters. The networks without self-attention are therefore referred to as $\text{SVNet}_{\text{DL}}(z_i)$. To show the superiority of the MVNet over SVNet, the MVNet is trained once with self-attention ($\text{MVNet}_{\text{DL}}(z_i)$) and once without self-attention ($\text{SVNet}_{\text{DL}}(z_i)$).

Additionally, to show that the use of self-attention is superior to multi-view consistency (Peters et al., 2020), another ablation study is performed in which SVNet is trained on a consistent label list $y_i^*$ instead of $y_i$. To create $y_i^*$ the majority label in $y_i$ is propagated to all unlabelled entries in $y_i$. This means that here, instead of correlating the input using self-attention, a single-view network is trained on all images, but the labels are propagated to all unlabelled multi-view images. In this way, the most frequent label is assigned to all related image pixels without replacing ground truth labels that do not match the majority decision. A single-view network trained on $y_i^*$ could learn a similar representation as a Multi-View Network trained on $y_i$, because $y_i^*$ is dense and all multi-view images are labelled.

In summary, three trials are performed: First, the MVNet that learns to relate the multi-view data using self-attention. Second, training a single-view network on the same data were nothing is related, and finally, training a single-view network were the majority label is propagated to all unlabelled list entries. In this step the single-view network has more ground truth data as in the second step and therefore also more training data.

Finally, note that each **training procedure is deterministic**, meaning that regardless of the loss functions, the presence of the attention mechanism or label propagation, the networks are always initialized with the same weights and always receive the same mini-batch examples in the same iteration step. Dropout or Augmentations such image flipping are also deterministic, so the only difference between the different outcomes are the selected hyperparameters and the presence or absence of the self-attention mechanism.

### 7.5.4 Ablation Studies and Results

The grids in Figure 7.17 and 7.18 show the mIoU values for all ablation results using self-attention vs. no self-attention. Each grid contains the mIoU for different window sizes $w \in \{0,2,4,32\}$ and batch sizes $b \in \{8,16,32,64\}$. Figure 7.17 shows the results using the Tversky loss and Figure 7.18 shows the same tests but with Dice Loss. Each training was performed once using the single-view network $\text{SVNet}_{\text{DL}}$ (No Attention) shown in the grids on the left and once using the Multi-View Network $\text{MVNet}_{\text{DL}}$ (Self-Attention) shown in the grids on the right. The color map shows higher mIoU values in dark red and lower mIoU values in light red or white. For easier comparison, all colors in the grids in this section are globally scaled, i.e. the white color corresponds to the global minimum and the dark red color to the global maximum.

At first glance, it should be clear that the networks benefit greatly from the multi-view observations. The MVNet performed here better in every single trial. The highest mIoU of 0.785 was obtained by $\text{MVNet}_{\text{DL}}$ using the Dice Loss with $b = 16$ and $w = 2$. In contrast, the highest mIoU of $\text{SVNet}_{\text{DL}}$ is 0.735 using Dice Loss and hyperparameters $b = 8$ and $w = 4$. In terms of hyperparameters, the single-view networks suffer more from using higher batch sizes. The Multi-View Networks are stronger regardless of batch size. This shows that $\text{MVNet}_{\text{DL}}$ should be easier to tune as it does not react as strongly to different hyperparameters.

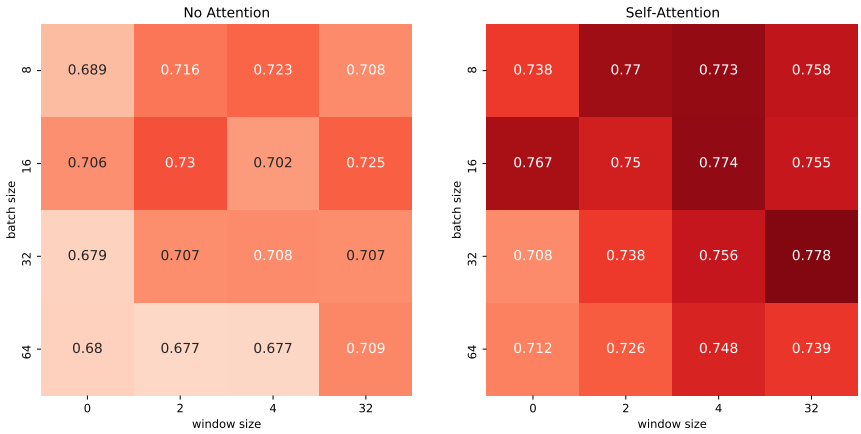



*Figure 7.17: Two grids showing the mIoU for the models using self-attention vs. no self-attention. The networks were trained using the* ***Tversky loss*** *with $\alpha = 0.5$ and $\beta = 0.5$ with different batch (rows) and window sizes (columns).*

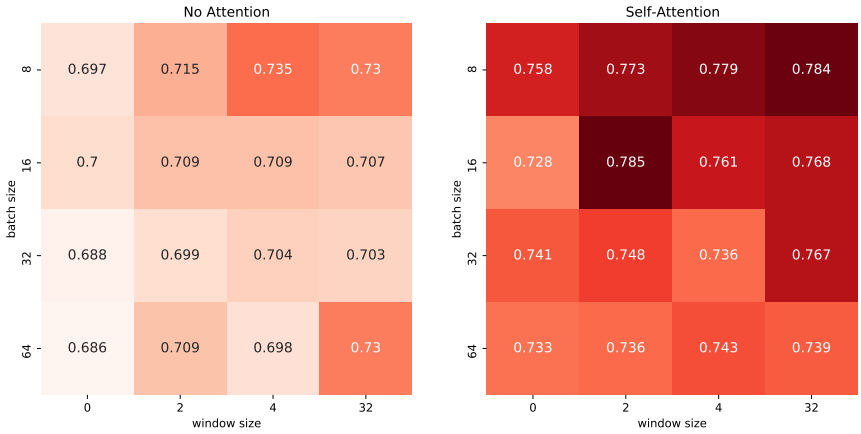



*Figure 7.18: Two grids showing the mIoU for the models using self-attention vs. no self-attention. The networks were trained using the* ***Dice loss*** *with different batch (rows) and window sizes (columns).*

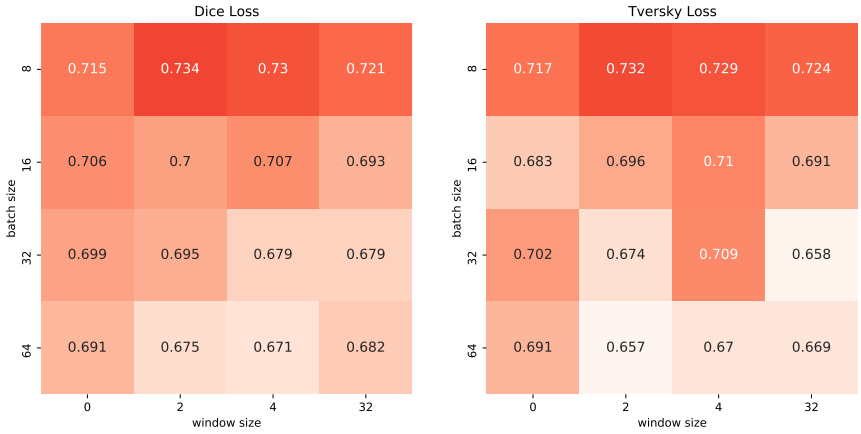



*Figure 7.19: Two grids showing the mIoU for SNet trained on the propagated labels (trial three). These models did not use self-attention. All ablations were done once for Dice and once for Tversky loss.*

A special case in the ablation study is when the window size is $w = 0$, which means that each network only has access to the predictions from Deeplabv3+. The results for the single-view networks all show that the predicted class distribution does not contain enough information to make predictions better than the Deeplabv3+ baseline. This would only be possible if there was any systematic effect in the class distributions, which apparently is not the case. On the other hand, MVNets are clearly superior in this respect, suggesting that linking multiple predictions for the same object is helpful for correcting errors. Looking at the columns where $w \neq 0$, it can be seen that the RGB data does increase the IoU for the single-view networks. However, the increase is still below the performance of the MVNet without RGB data. This is best seen in Figure 7.18, where most single-view networks that have access to RGB information lag behind Multi-View Networks without RGB information. The fact the RGB information in itself improves performance becomes clear when comparing $w = 0$ with $w > 0$.

Finally, the grids in Figure 7.19 show the results when the single view networks are trained on the labels propagated through all the multi-view-images. These results are interesting as they complete the validation of the whole approach. It can be seen that no significant advantage is obtained over the pretrained model, neither by using the naive approach nor by linking the labels. It is only by relating the inputs using self-attention that a significant improvement can be measured that did not appear anywhere else in all the ablation studies. Apart from this, the grids in Figure 7.19 show that with larger window size and batch size the results tend to be slightly worse. One reason for this could be that the propagated labels generate label noise due to occlusions and calibration errors. Using only the geometric correspondence between camera poses and 3D points cannot account for these issues, and could affect the overall performance of the network.

#### 7.5.4.1 Analysis of the Effect on Predictions Using Multiple Views

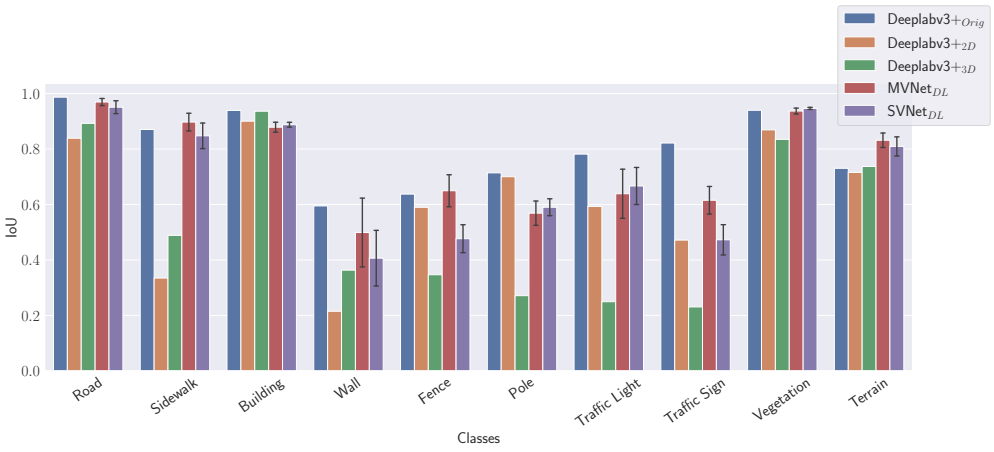



*Figure 7.20: Comparison of $MVNet_{DL}$ and $SVNet_{DL}$ with the baseline shown in Figure 7.2. The height of each bar for each class is given by the mean across all experiments shown in Figure 7.18 and 7.17. The black error bars show the standard deviation for each class within all ablation experiments. Please note the results for $Deeplabv3+_{2D}$ are obtained using the total reference set and are therefore not directly comparable to the ones of $MVNet_{DL}$ and $SVNet_{DL}$. They are intended to show in which cases label transfer from 2D to 3D lead to an increase or decrease of IoU. The actual improvement to the Deeplabv3+ baseline is shown in the next subsection.*

At the beginning of Chapter 7.1, the baseline for label transfer from segmented images to 3D points with Deeplabv3+ was presented. In Fig. 7.2, it was shown that Deeplabv3+ suffered from a domain gap in the MMS images, especially for the *sidewalk* and *wall* classes. Interestingly, in these cases, the aggregation of labels by multiple 2D observations for a single 3D point resulted in a higher mIoU

in the 3D dataset than in the 2D MMS-dataset, suggesting that multiple observations were helpful in mitigating label noise by majority voting over multiple predictions. Here, the network does not propagate a label to a 3D point, but has access to exactly the same multi-view observations. It would therefore be useful for the $\text{MVNet}_{\text{DL}}$ network to be able to use the multi-view observations in the same way, by learning to propagate individual predictions through space and assigning correct labels to pixels where Deeplabv3+ suffered from the domain gap. In the best case, $\text{MVNet}_{\text{DL}}$ can learn to correct wrong predictions for, say, *sidewalk* and *wall* by using the (correct) predictions from other views, while keeping the correct predictions in all other cases. Additionally $\text{MVNet}_{\text{DL}}$ can make use of the provided features like RGB data and the initial Deeplabv3+ predictions to learn how to correct single-view predictions.

The barplot in Figure 7.20 shows that this is the case. The blue bars show the class-wise IoU on the original dataset (Cityscapes). The orange bars show the estimated IoU for the MMS-dataset, where the difference between then indicates the domain gap. The green bars show the estimated mIoU on the 3D dataset, i.e. after the label transfer. Finally, the red and violet bars show the result for each class for the multi-view and single-view networks. These results were created by averaging all results that where conducted in the ablation studies, therefore the black bars indicate the variance of each result. At first glance, it can be seen that $\text{MVNet}_{\text{DL}}$ performs significantly better than the single-view model in many cases, especially for the *sidewalk*, *wall*, *fence*, *traffic sign* and *terrain*.

#### 7.5.4.2 Improvement Compared to the Baseline

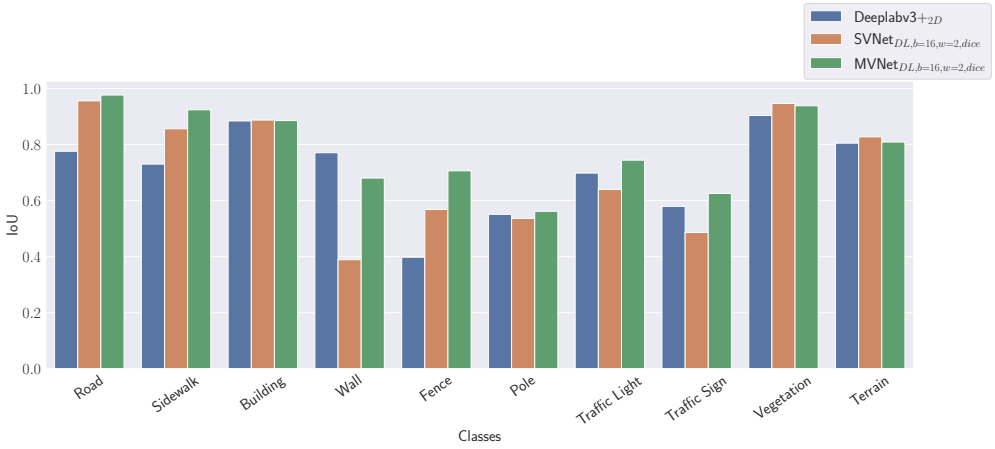



*Figure 7.21: Comparison of the baseline result for Deeplabv3+ before and after correction by SVNet or MVNet, both using the same hyperparameters. All results were obtained using the same test set.*

To show the actual improvement on the MMS dataset, the results for $\text{MVNet}_{\text{DL}}$ are compared to the baseline predictions of Deeplabv3+ on the MMS dataset. For the Multi-View Network, the best performing model was selected ($w = 2$, $b = 16$ with Dice loss). For reference, the single-view network trained with the same hyperparameters is also shown. In all cases, the mIoU was calculated using the test set presented in Section 7.5.1. Note that the performance for Deeplabv3+$_{2D}$ was also calculated using the same subset to allow a fair comparison. The Figure 7.21 show that the Multi-View Network improved the IoU in almost all classes. In terms of numbers, the mIoU increased from 0.709 (baseline) to 0.785 ($\text{MVNet}_{DL}$). In contrast, the single-view counterpart only achieved an mIoU of 0.709 and 0.735 in the best case, showing the significant improvement due to the Multi-View Networks. Interestingly, the Multi-View Network did not perform better in all classes compared to the Deeplabv3+ baseline, see Fig. 7.21. It can be seen that for the class *wall* the

baseline is slightly better. To compensate for this, the solution is quite simple. For classes where the original predictions are better, a set of classes $K$ is defined where the baseline predictions are kept and are not replaced by the Multi-View Network. The following function $M(\hat{y}_{DL},\widetilde{y}_{\mathrm{MV}})$ uses two corresponding predictions as input, one by the baseline ($\hat{y}_{DL}$) and one by the multi-view correction network ($\widetilde{y}_{\mathrm{MV}}$) and merges them in the following way:

$$M(\hat{y}_{DL},\widetilde{y}_{\mathrm{MV}}) = \begin{cases} \hat{y}_{DL}, & \text{if } \widetilde{y}_{\mathrm{MV}} \in K \wedge \hat{y}_{DL} \in \text{static} \ . \\ \widetilde{y}_{\mathrm{MV}}, & \text{otherwise.} \end{cases} \tag{7.10}$$

Here static defines the set of all static classes, see Table 6.2. This means that the function $M$ returns the base model prediction wherever the multi-view model is known to be worse, as long as the base model prediction is part of the supported set $S$. By using the Equation 7.10 with $K = \{\text{wall}\}$, the mIoU increases to 0.802, which is a good improvement over the original MVNet$_{DL}$, which reached 0.785.

### 7.5.5 Qualitative Evaluation

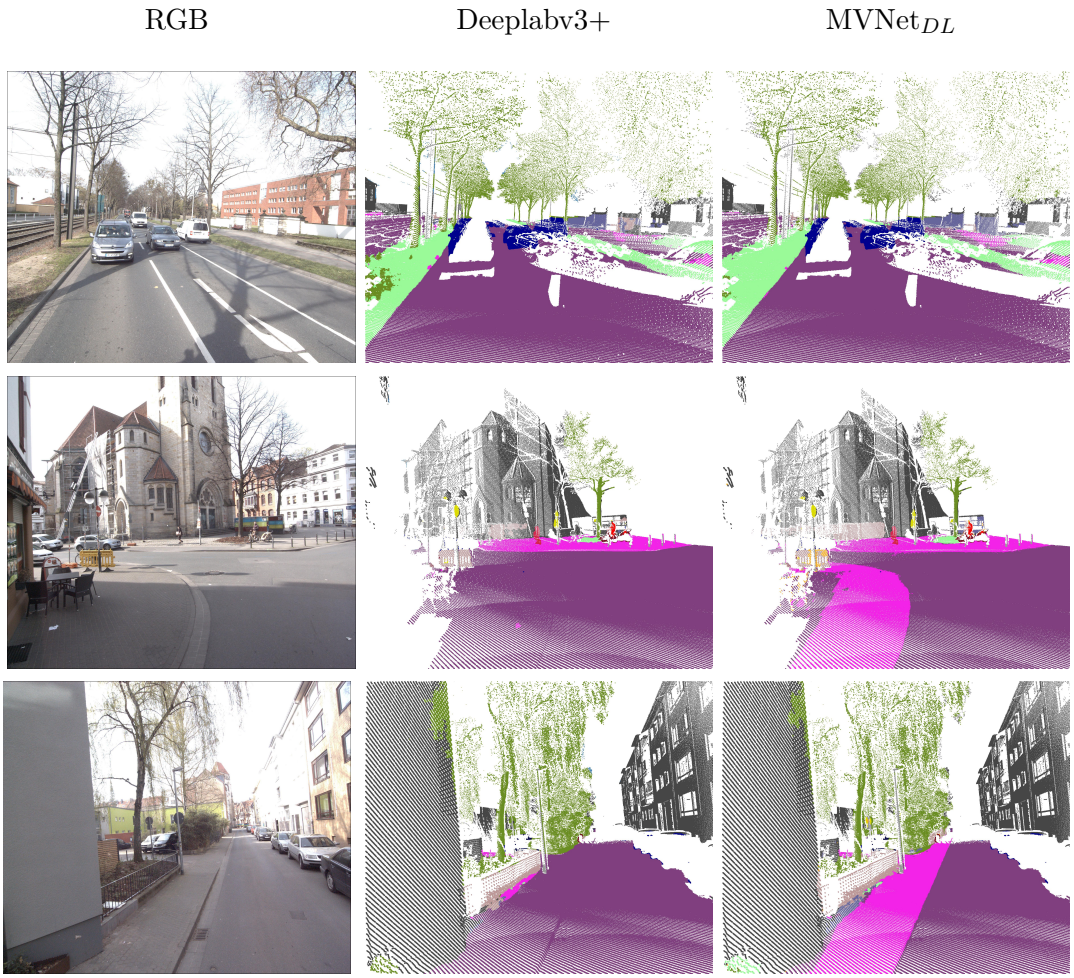



*Figure 7.22: Qualitative comparison of the original predictions by Deeplabv3+ (middle) and the predictions by MVNet$_{DL}$. The left column shows the original images. The column in the middle shows the predictions from Deeplabv3+ if a 3D point is available. The right column shows the corrected predictions by MVNet$_{DL}$*

For qualitative evaluation, the prediction $\widetilde{y_i}$ by $\text{MVNet}_{\text{DL}}(z_i)$ are shown in the respective image. For better visibility, the drawing size of each pixel was increased depending on the distance between the 3D point and the camera center.

The images in Fig. 7.22 show some examples before and after correction in 2D by $\text{MVNet}_{DL}$ without using Equation 7.10 to merge the results. The images are generated by drawing only pixels in which a 3D point is associated with the corresponding 2D pixel, or in other words, if the point cloud were projected into the respective image, the 2D predictions are shown only if a 3D point would also be visible. However, as the Multi-View Networks can only correct static classes in the MMS image dataset, it is unclear what to do with pixels that belong to a dynamic class according to Deeplabv3+. If a pixel is associated with a 3D point and Deeplabv3+ has predicted a dynamic class, the dynamic class simply replaces the prediction with $\text{MVNet}_{\text{DL}}(z_i)$ and is drawn into the image instead. This is the reason why cars or pedestrians can be seen to some extent in the images shown.

The first thing to notice in the pictures is that the missing sidewalk (pink) could be restored in most of the pictures. On the other hand, correct predictions were not changed. Fig. 7.22 shows that in the first row in the background the detected *fence* was correctly changed to *wall*. Similarly the wrongly detected *vegetation* on the ground is altered to *terrain* (light green). The second and third rows are showing two restored sidewalks which were not available in the original Deeplab predictions. Additionally in the second row the *fence* and *terrain* in the background around the church are correctly predicted by $\text{MVNet}_{DL}$.

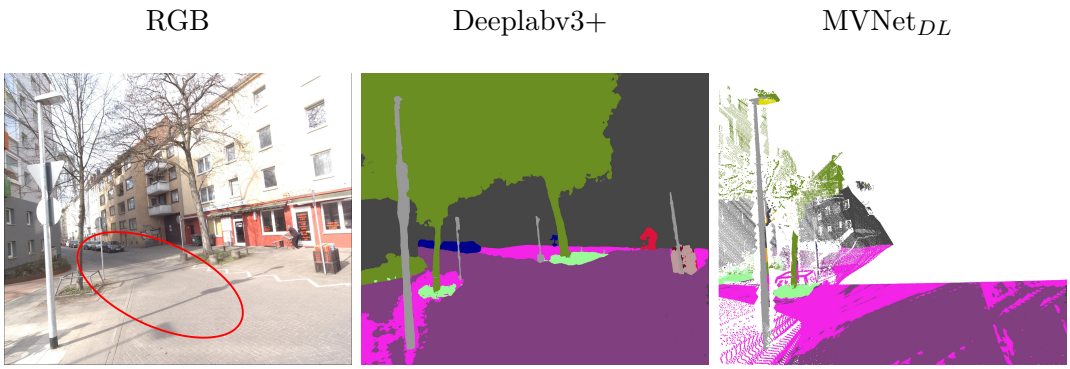



*Figure 7.23: An example of an error case for MVNet. The first image shows a traffic-calmed road (red ellipse), correctly classified by Deeplabv3+ (middle image) as road (purple) and by $\text{MVNet}_{DL}$ (right image) predominantly as sidewalk (pink).*

In some rare cases the Multi-View Network fails to predict the correct class as shown in Fig. 7.23. in this example, MVNet classified most of the road as a sidewalk, which is obviously wrong.

### 7.5.6 Retraining Semantic Segmentation Networks

In this section, several studies are performed to measure and compare the fine-tuning of Deeplabv3+ on the MMS data. In the **main study**, Deeplabv3+ is fine-tuned on a dataset with 2636 image pairs and labeled images by $\text{MVNet}_{DL}$. The labels are created by using the merge strategy according to Equation 7.10. Figure 7.24 shows some randomly selected examples from this dataset. Fine-tuning is done by training Deeplabv3+ for 20 epochs using the Adam optimizer with the default parameters, i.e., a learning rate of $\lambda = 0.001$ and $\beta_1 = 0.9$ and $\beta_2 = 0.999$. However, the learning rate halves every 5 epochs if the validation mIoU has not increased. Since the xception backbone is computationally very expensive, the batch size is set to 4 and the image size is reduced



*Figure 7.24: Randomly sampled pairs of images and labelled images created by MVNet using the merge strategy from Equation 7.10. The RGB images show the input data and the images to the right shows the predictions by MVNet which are used as pseudo labels for fine-tuning of Deeplabv3+.*

by 0.6, i.e., it is set to $1233 \times 1471$ pixels which are the maximum values a single Nvidia Titan X with 12GB can handle for training. To prevent overfitting of the network, the backbone of the network is frozen and only the decoder part of Deeplabv3+ is trained. This is also done in the ablation studies. This trial will be referred to as $DL_{\mathbf{MV}}$ in the following.

To evaluate and rank the fine-tuning results, **three ablation studies** are performed. **First**, the pretrained Deeplabv3+ network is fine-tuned directly on the training data that was available for MVNet using almost the same hyperparameters as above. It is expected that the network will overfit as the training set contains very few reference labels. Therefore, the learning rate is reduced to 0.0004 and an additional weight decay is added with a factor of $10^{-6}$. Since this study relies only on supervised fine-tuning, it is referred to in the results as the **Naive** Approach.

The **second ablation** study will use a self-training approach. Similar to the work of Zhu et al. (2020), Deeplabv3+ will be jointly fine-tuned to pseudo-labels and the training set used for training MVNet using a student-teacher approach. The pseudo-labels are generated using the pretrained "teacher" DCNN on the same 2636 images as in the main study. For a fair comparison, the approach by Zhu et al. (2020) is slightly adjusted, but still follows the general strategy. The "student" network is the same network as in first trial and uses a frozen encoder. Instead of stochastic gradient descent, the same optimizer with the same hyperparameters in the main study is used. Also instead of training on cropped images, the student is trained on the same images as in the main experiment. In the original implementation, Zhu et al. (2020) merged the sets of reference

images and pseudo-labelled images into one training set from which random samples are drawn. A similar approach is taken here, but since the training set contains only 17 images, the ground truth images are shown twice per epoch, once in their original form and once flipped horizontally (left-right flipped). This study is referred to as **self-trained** in the following results.

In the **third ablation**, $DL_{\mathrm{MV}}$ is extended to include the self-training paradigm from the previous ablation study. Here Deeplabv3+ is jointly fine-tuned on the corrected images from MVNet and on the sparse ground truth data. Apart from this, nothing is changed to the main study. This study is referred to as $DL_{MV,ST}$.

To make the studies fair, all ablation studies containing human-annotated ground truth data (naive, self-trained, $DL_{\mathrm{MV}}$ and $DL_{MV,ST}$) are trained on exactly the same labels that MVNet was trained on (see Figure 7.16 "Associated train set"). Please note that the validation and test sets are the same in all studies (see Figure 7.16 "Associated validation and test set"). As no labels for dynamic objects were used in the training process of MVNet, the labels for dynamic objects from the "leftover validation set" (see Figure 7.16) are also included in the validation set, which were not previously used in any other study or trial. Finally, testing is done on the "leftover test set"(see Figure 7.16) which contains labels that neither MVNet nor any other network has previously seen or been trained.

### 7.5.7 Results of the Retraining Process

Qualitative results before and after fine-tuning are shown in Figure 7.25. The pretrained network without any fine-tuning shows relatively good results. The most striking error is that *sidewalk* is often confused with *road*. For example in the last row of Figure 7.25, Deeplabv3+ merely recognized the sidewalk on the left and the island in the left center of the image. Similar cases can be seen in the third row and fourth row on the left of the images. The same applies for the class *terrain* that was sometimes confused with *vegetation*, for example in the fourth row on the right. Using self-training labels only improved this problem slightly and even when the network was trained directly on the ground truth the sidewalks still has some gaps in the predictions. The network that was fine-tuned on the corrected dataset by MVNet has almost none of these errors.

| mIoU / Method | Val mIoU | Test mIoU (static classes) | Test mIoU (dynamic classes) | Test mIoU |
|---|---|---|---|---|
| Deeplabv3+ (pretrained) | 0.611 | 0.674 | 0.492 | 0.632 |
| Naive | 0.767 | 0.705 | 0.374 | 0.628 |
| Self-Trained (Zhu et al., 2020) | 0.638 | 0.691 | **0.533** | 0.663 |
| $DL_{\mathrm{MV}}$ | 0.701 | 0.749 | 0.516 | **0.696** |
| $DL_{MV,ST}$ | 0.719 | **0.753** | 0.492 | 0.692 |

*Table 7.7: Table with all the results for the individual ablation studies. The mIoU is given for the validation (Val mIoU) and testing sets (Test mIoU). In addition, the test MIoU is also shown separately for static classes (third column) and dynamic classes (fourth column).*

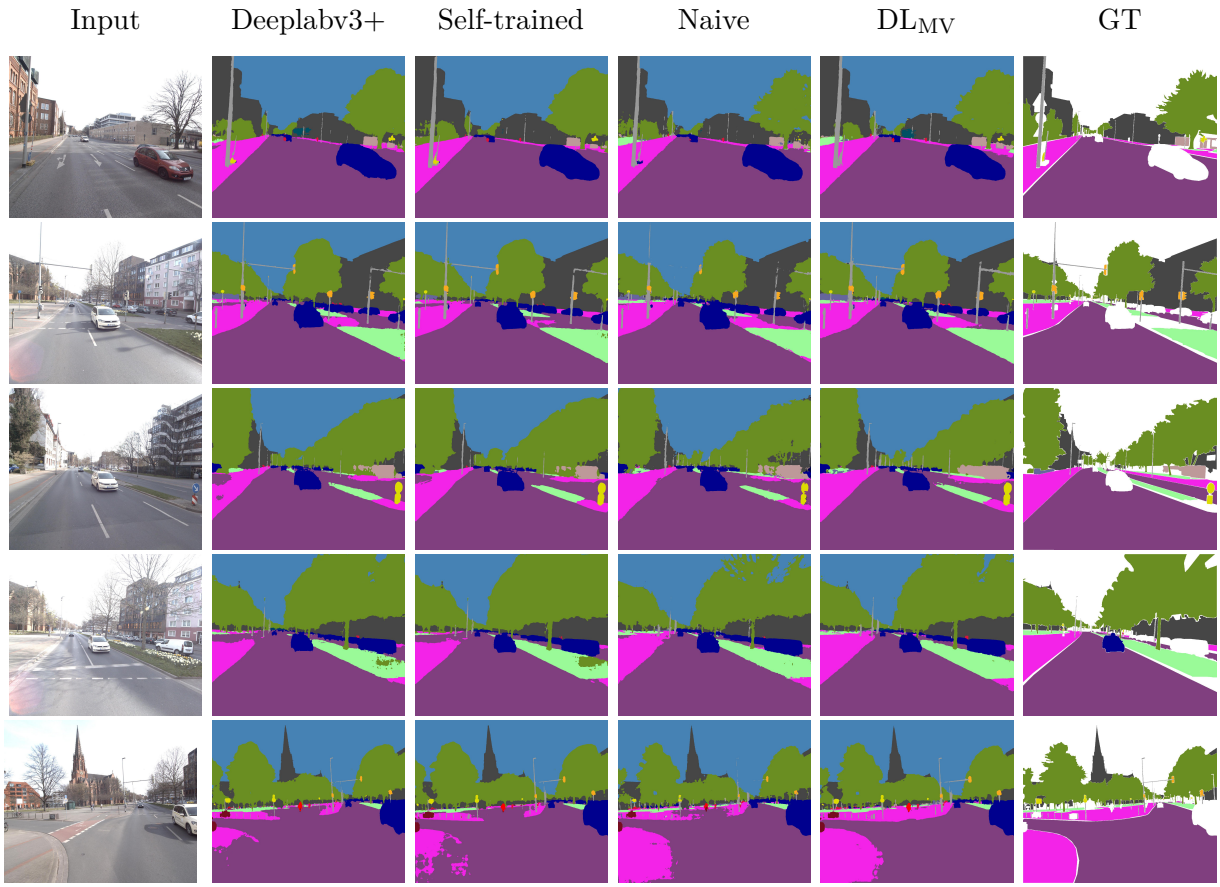



*Figure 7.25: Results on the "leftover test set" (see Figure 7.16). The first column shows the input and the last column shows the ground truth data. The columns Deeplabv3+ show the pretrained model, Pseudo shows the model after training with pseudo-labelling and fine-tuned shows the pretrained model after training directly on the train-set. $DL_{MV}$ shows the fine-tuned model on the corrected dataset by MVNet.*

Table 7.7 shows the results on the training, validation and test sets for all studies. The evaluation was performed separately for dynamic and static classes. As no labels for dynamic objects were used in the training process, it is expected that the fine-tuned DCNN will mainly improve the classification performance for static classes. The performance for dynamic classes is nevertheless interesting, as the network should not deteriorate here after fine-tuning. As Baseline the pretrained Deepblav3+ achieves an mIoU of 0.63 for all classes and 0.67 for static classes. With naive fine-tuning on the training set, the network performance is slightly better (second row) for static classes. However as the dynamic classes deteriorated with naive fine-tuning the test mIoU actually got worse. Using self-training (third row) significantly improved the performance for dynamic and slightly for static classes. When comparing the validation mIoUs with the test mIoUs it can be seen that by far the biggest gap occurs when the network was naively fine-tuned on the reference set, suggesting that this network overfitted to the data. Overall, self-training improved the test mIoU from 0.63 to 0.66 using only very few labels. The best results are achieved by the DCNN fine-tuned on the pseudo-labels from MVNet. Here the mIoU increased significantly for all static classes from 0.67 to 0.75.

In Figure 7.26 all IoU values per class are visually compared. It can be seen that the network trained on the corrected MVNet dataset achieves a very good result almost everywhere. The bar chart also supports the findings from the qualitative evaluation that Deeplabv3+ performs best on the classes *sidewalk* and *terrain* when trained on the corrected dataset. Finally, Figure 7.27

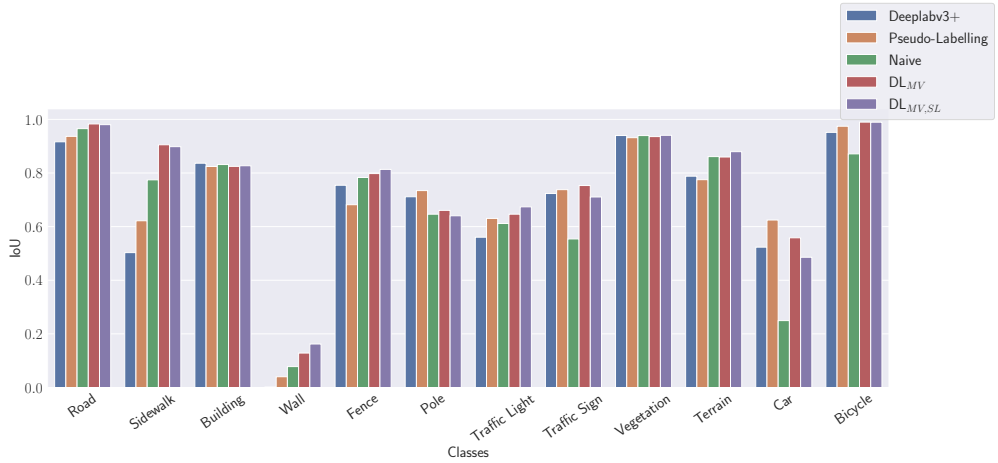



*Figure 7.26: Comparison of the IoU per class for all ablation studies. Note that class person is not included in the test set.*

shows some randomly drawn results from the MMS image dataset that were not part anywhere in training, validation or testing subsets. It compares the predictions of all experiments from different locations that have not yet been used in the training processes. These image should further support that the predictions of the fine-tuned Deeplabv3+ network are very consistent and that the network has improved its overall performance compared to the others.

### 7.5.8 Conclusion and Discussion

This chapter had two objectives, the **first** was to evaluate the Multi-View Network as a tool for dealing with unordered, multi-view observations of variable length. These networks are able to relate and extract clues from corresponding multi-view images. It was shown that using the Multi-View Network significantly outperformed the corresponding Single-View Network. Even when the network only had access to the prediction of each central pixel, it was able to use the relationship between all of them to achieve a higher mIoU than most other Single-View Networks that had access to even more features. It is assumed that the reason for the higher performance lies in the ability to relate different predictions.

The **second** goal was to show that the Multi-View Network generalises well enough that its prediction can be used to fine-tune a DCNN in the new target domain. Of course, the DCNN can be fine-tuned directly using the target reference data. But as studies have shown, the DCNN easily overfits to the training set since it only contains the amount of labels of three to four images. Even though the pretrained DCNN performed relatively well on the MMS-dataset the network increased its performance by 8 % for static classes.

Further improvements of the Multi-View Network are up to discussion. The network might benefit from semi-supervision, as the Scanstrip Network did in Section 7.4.3. By pretraining the Multi-View Network with the predictions of the DCNN as pseudo-labels and later exchanging them with the reference labels in the fine-tuning step. It is also conceivable to convert a pretrained DCNN into a Multi-View Network and thus use transfer learning to achieve better performance. Finally since deep learning methods are known to require a lot of training data, it would be interesting to see how the network performs on a much larger multi-view dataset. Here it would be interesting to see if the Multi-View Network is able to reduce the known error cases and maintain the performance gap to corresponding single-view networks.

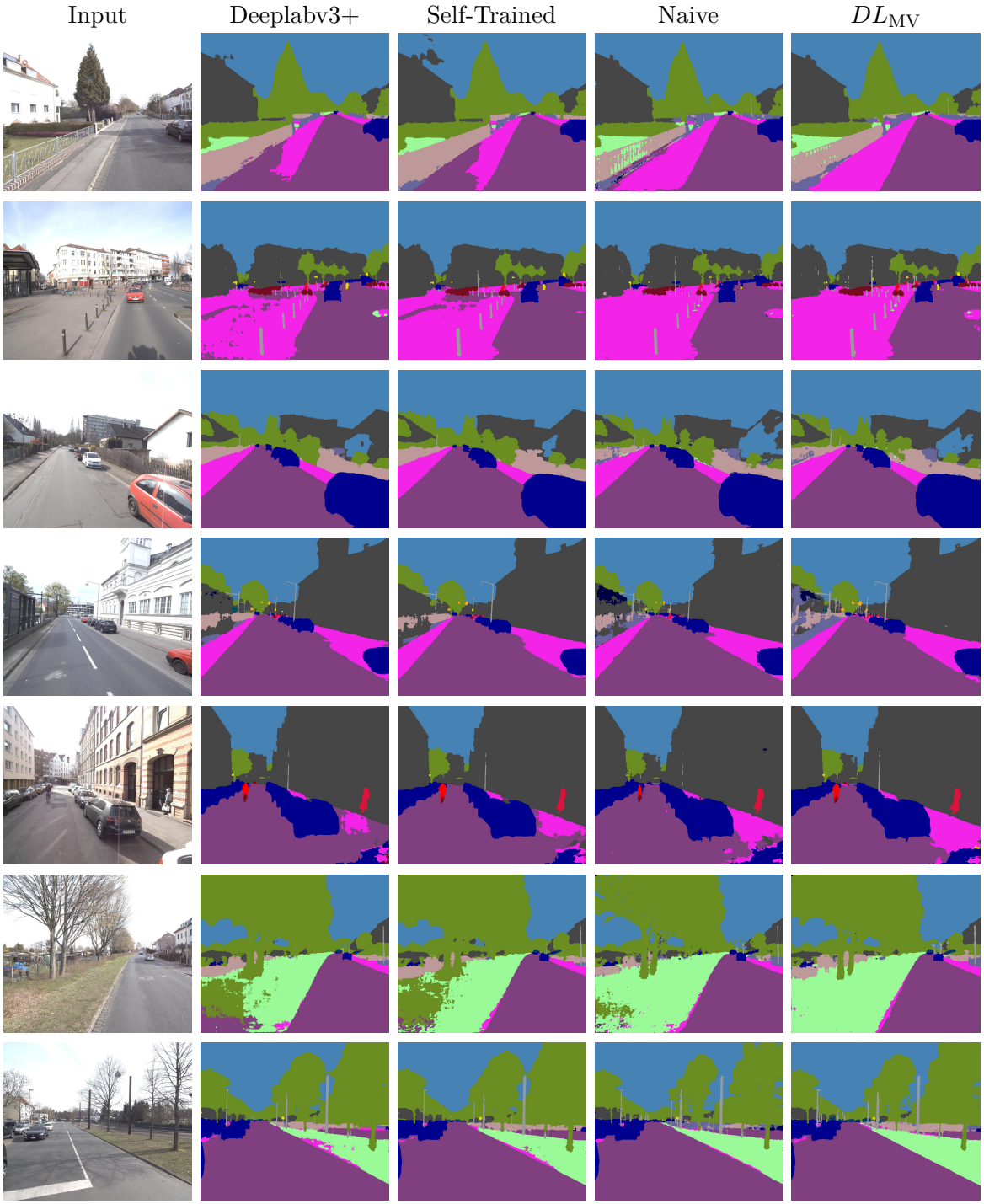



*Figure 7.27: Randomly selected images from the MMS dataset except for the training, validation and test set. The first column shows the input for the networks. The columns Deeplabv3+, Self-Trained, Naive, and $DL_{MV}$ show the respective experiments.*

## 7.6 Multi-View Label Transfer Learning

As seen in Table 5.2, the previous sections have shown how to deal implicitly or explicitly with error causes in label transfer. Each of the sections has shown how to improve over the previously shown baseline by filling the domain gap in the target dataset or dealing with different types of occlusions in the mapping process. In this section, all the methods shown so far are combined to learn label transfer end-to-end. As explained in Section 4.3.2, the Label Transfer Network (LTNet) works by combining the MVNet from the previous section with the Scanstrip Network from the first section so that it has access to 3D and multiple 2D observations simultaneously. The 2D observations allow the LTNet to learn how to suppress wrong predictions by the pretrained DCNN in order to assign a correct label to the corresponding 3D point. By observing local 3D structures using the SNet generated features, the network can learn about different types of occlusions, calibration or label errors.

As shown in Fig. 4.14, LTNet is trained using the 3D reference set. In this section, the creation of the training ,validation set is skipped as this approach uses exactly the same sets as presented in Section 7.3 to make it directly comparable to the previous results obtained for SNet and $\text{SNet}_{SSL}$.

### 7.6.1 Training Procedure

First, the notation of $\text{LTNet}(z_j,v_j,\Omega(\text{SNet}_{SSL}(S_j),n-1)_j)$ from Section 4.3.2 is briefly repeated. The network LTNet consists of two subnetworks, a 2D multi-view branch and a scanstrip branch, see Fig. 4.14. The multi-view branch receives the lists $z_j$ and $v_j$ which contain the multi-view image features and the 3D point features corresponding to the 3D point $p_j$. The 3D branch is given by the the feature vector $\Omega(\text{SNet}_{SSL}(S_j),n-1)_j)$ which is extracted from a scanstrip $S_j$ using the pretrained network $\text{SNet}_{SSL}$ from Section 7.4.3. For tuning the LTNet a grid search was performed with the following hyperparameters: Similar to Section 7.5, the batch size $b \in \{8,16,32\}$ and the window size $w \in \{0,8,32\}$ were tuned. In addition, each trial was conducted with and without the attention mechanism. To compare the influence of both parts (the multi-view part and the scanstrip network), each trial was also run with only the Multi-View Network $\text{LTNet}(z_j,v_j)$ and only the scanstrip features $\text{LTNet}(\Omega(\text{SNet}_{SSL}(S_j),n-1)_j)$. To be more precise, both subnetworks, the Multi-View Network part and the scanstrip part, extract a fixed size feature vector, which are concatenated and passed together to two successive fully-connected layers that return the final prediction, see Fig. 4.14. In the first case, where the 3D features are not available, the scanstrip feature vector is not passed to the fully-connected layers. In the other case, where only the scanstrip feature vector $\Omega(\text{SNet}_{SSL}(S_j),n-1)_j$ is available, the Multi-View Network is not concatenated to the fully-connected layers. Finally, all trials were performed with cross-entropy as loss function, resulting in a total of 30 different trials. In all cases the features $\Omega(\text{SNet}_{SSL}(S_j),n-1)_j$ were extracted from $\text{SNet}_{SSL}^{(256)}$, which is described in Section 4.2.2. Every model was trained for 50k iteration using Adam optimizer with standard parameters ($\lambda = 0.001,\beta_1 = 0.9\beta_2 = 0.999$). For multi-view observations only camera images within a range of 40m from the 3D point were considered. Similar to the previous ablation studies, care was taken to ensure that all trials were deterministic, meaning that the weights were initialised with the same values in all cases and the networks received the same mini-batch samples as long as they had the same hyperparameters. Aspects such as augmentation (random image flipping) and dropout were also deterministic.

A Problem that has not yet been addressed is the **complexity of the input data structure**. Due to the multi-view inputs, the network needs to have access to different image patches, image predictions and 3D features at the same time. For training the LTNet, this means that the network needs access to 1519 different images and another 686 images for testing. The training images alone consume about 23 GB of memory and the corresponding predicted class distributions (a vector

of length 19 for each pixel) from Deeplabv3+, stored as 32bit floats, require about 582 GB of memory. In addition, the features for each point in 3D are collected in a vector of length 64, stored as 32-bit floats and requiring around 45 GB of memory. Traditionally, in most frameworks, the images are preprocessed in parallel as mini-batches and added to a queue from which the main loop retrieves the batches for training. In this case, that means that for a single sample for LTNet, these data generators would have to load multiple corresponding images, make predictions for the central pixels, and load the corresponding 3D feature for the 3D point. As it is difficult to estimate in advance which images will be loaded, because mini-batches should be sampled randomly, this process is very time consuming. There are many ways to solve this problem. First, all the data can be preprocessed once and stored on disk, which (1) consumes a lot of HDD memory because some data is stored redundantly, and (2) the data must be processed each time for different hyperparameters such as the size of the input window. The second solution stores only the pointer to the data and the raw input data, so the batches are created on the fly. This solution is the slowest, but uses the least memory. Here, a hybrid solution was chosen, where the data that does not change depending on the hyperparameters is preprocessed and stored in a hierarchical data format (HDF5), and other data such as the input images are stored as pointers, so that the image patches can be looked up on the fly. To be sufficiently fast, this means that all images were kept in memory. Still, the training and overall inference speed is something that should be addressed in future work and will be discussed in the conclusion.

### 7.6.2 Ablation Studies and Results

Figures 7.28 and 7.29 show the results of the grid search for hyperparameter tuning of LTNet. In Section 7.4.3 $\text{SNet}_{SSL}^{(256)}$ achieved an mIoU of 0.71. A look at the 7.28 Figure shows that the LTNet performed better in almost all cases than $\text{SNet}_{SSL}^{(256)}$. The general result and the influence of the self-attention mechanism are discussed in the following subsection. The influence on those results with respect to the individual branches of the network, i.e., the impact of the 2D and 3D features, is discussed in the section after that.

#### Impact of Self-Attention

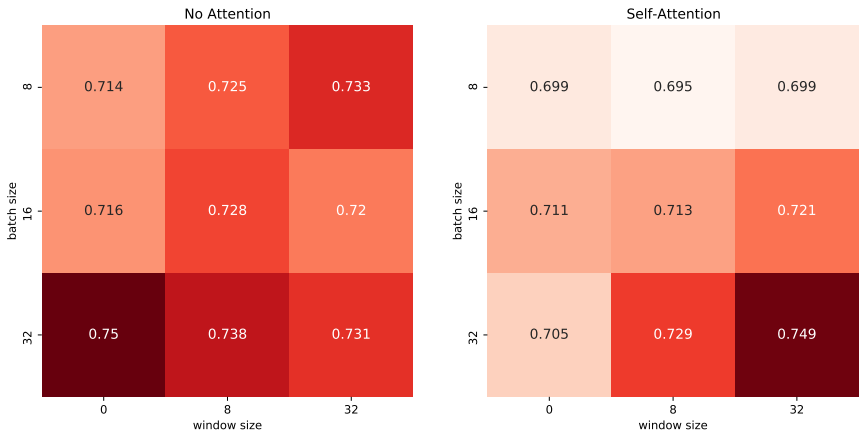



*Figure 7.28: mIoU on the test set for LTNet using no attention (left) or self-attention (right) in the multi-view branch. The networks were trained using* ***cross-entropy*** *with different batch (rows) and window sizes (columns).*

Comparing both grids in Figure 7.28 shows the impact of the self-attention mechanism in the multi-view subnetwork. The network with no attention, a batch size of 32, and a window size of 0

performed best by a very small margin. However, comparing the two grids, the best performance is very similar. This raises the question of whether self-attention is necessary in this network design, since it did not significantly increase the mIoU. It is important to note that this finding does not contradict the previous results from the Multi-View Networks. Although both models, MVNet and LTNet, have similarities in their network design, the main difference in LTNet is that all multi-view feature vectors are aggregated into one fixed-size feature vector using average pooling. Apparently, the use of self-attention and thus the linking of all observations before aggregation does not provide an advantage as significant as in Multi-View Networks.

### Impact of 2D and 3D Features

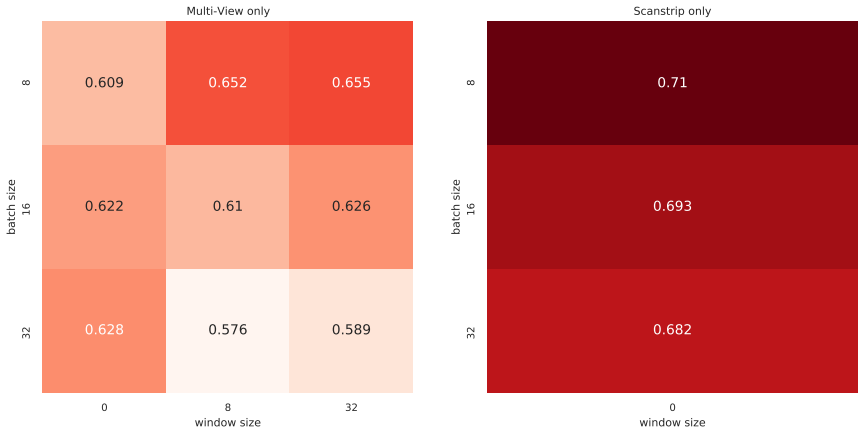



*Figure 7.29: Two grids showing the test-mIoU for the models using only multi-view (2D) features vs using only the scanstrip feature vector (3D). The networks were trained using cross-entropy with different batch (rows) and window sizes (columns).*

The grids in Figure 7.29 compares results for subnetworks having only access to either the multi-view data (2D) or to the feature vector extracted from $\text{SNet}_{SSL}^{(256)}$ (3D). As expected, the network that only has access to the scanstrip feature vector performs very much like the original $\text{SNet}_{SSL}^{(256)}$. The reason for this is that no additional information has been added and therefore the performance depends on the loss that move around the local optimum. More interesting is the grid on the left side of Fig. 7.29. This grid shows that if the network relies almost entirely on 2D features for label transfer, it will not be able to obtain an mIoU above 0.655 ($\text{LTNet}(l_i,\hat{y}_i,r_i)_{w=32,bs=8}$).

To analyze this further, two confusion matrices are shown in Fig. 7.30. The matrix on the left shows the confusion when the network has **no access** to the 3D features by $\text{SN}_{SSL}^{(256)}$ and the matrix on the right shows when the same network **has access** to those 3D features. The matrices are normalized per row, similar to the confusion matrix shown in the baseline. For Figure 7.30 the matrix on the left shows a very similar pattern to the one shown in the baseline, see Fig. 7.3 (right matrix). It is reasonable that LTNet without the 3D scanstrip features would create similar errors because it is hard to map the corresponding 2D image labels to a 3D point without having much information about the 3D point and its neighbors. Therefore, these errors can also be seen here, such as calibration errors with classes that have a small or thin physical size, such as *pole*, *traffic light* and *traffic sign*. These classes are often confused with surrounding objects such as street, sidewalk and building that are not semantically similar. Similarly, dynamic objects such as *person*, *car*, *truck*, and *bicycles* are confused with surrounding classes due to dynamic occlusions. However, as can be seen in the right confusion matrix, these problems become less severe when the 3D features from $\text{SNet}_{SSL}^{(256)}$ are also added (especially for dynamic classes) Not only is the overall

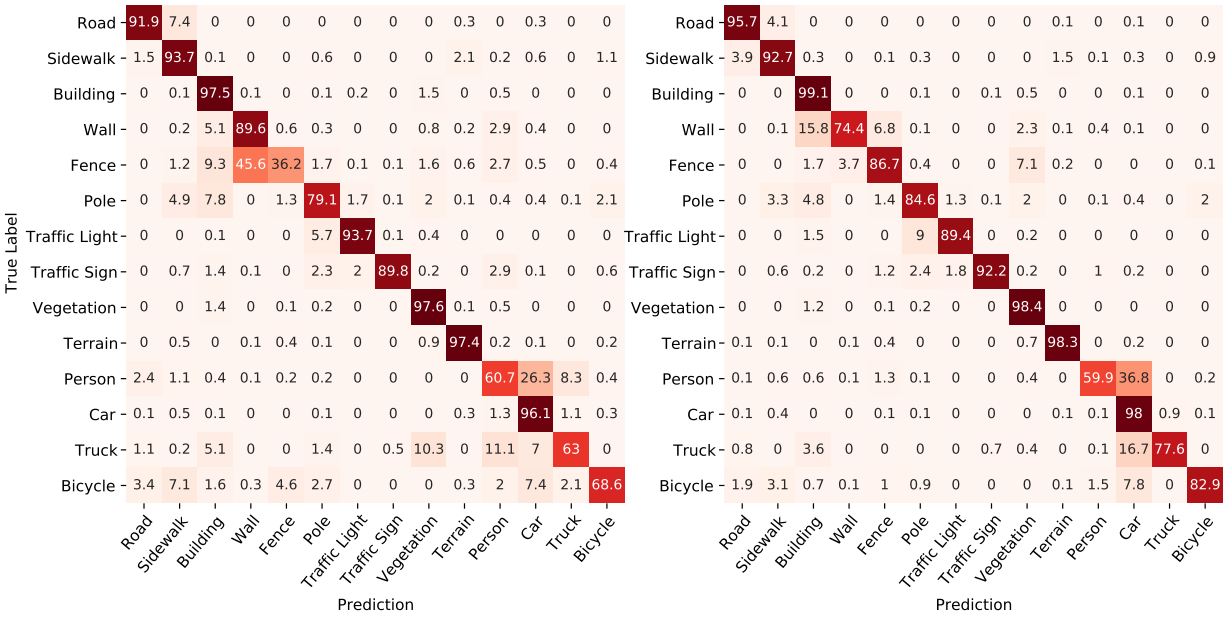



*Figure 7.30: Two confusion matrices for LTNet$_{w=0,bs=32}$. The left shows the confusion when only multi-view features (without 3D scanstrip information) are used. The right side shows the results when all features are used. The matrices are normalized per row.*

accuracy higher here, but there is also often less confusion with surrounding objects. This suggests that LTNet can directly account for these problem to map the correct label from 2D to 3D.

### Comparing the Results to Previous Methods

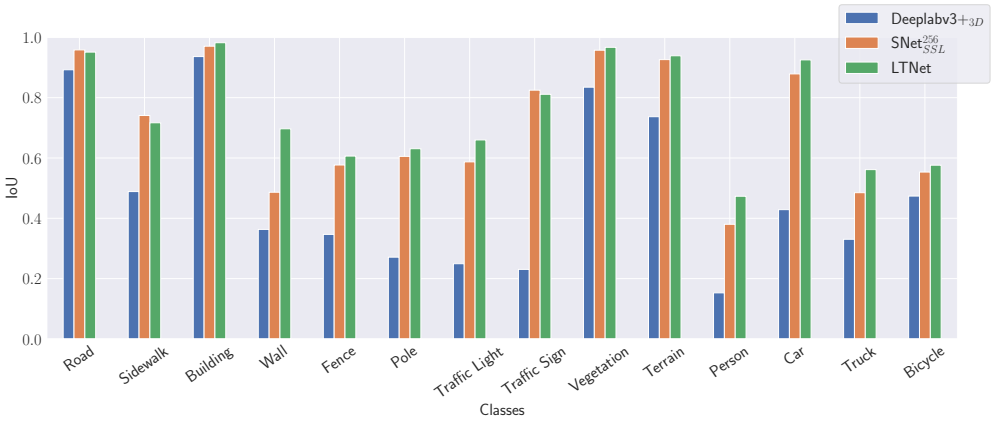



*Figure 7.31: Comparison of IoU per class between naive baseline (blue), semi-supervised scanstrip network (orange) and LTNet (green).*

Of all the ablation studies, LTNet achieved a best-case mIoU of 0.75. Compared to the other networks (see Table 7.5), this is significantly higher than all competitors. As this network used no additional reference data and exactly the same training, validation and test splits as the others, this is a fair comparison. The IoU per class is shown in Figure 7.31, comparing LTNet to the naive baseline and the former best network SNet$^{(256)}_{SSL}$. The Figure shows that LTNet outperforms its competitors for most classes. In particular, for the classes *wall*, *traffic light*, *person*, *car* and *truck*, the IoU is significantly higher.

*Table 7.8: Comparison of methods applied directly to the scanstrip and LTNet. All networks were trained, validated and tested on the same sets. The input patch size is* $256 \times 256$ *in all cases.*

| Method | SNet | $\text{SNet}_{SSL}^{(256)}$ | HRNet | FCN | U-Net | Deeplabv3+ | LTNet |
|---|---|---|---|---|---|---|---|
| mIoU | 0.665 | 0.709 | 0.628 | 0.549 | 0.630 | 0.620 | **0.750** |

### 7.6.3 Qualitative Evaluation

For qualitative assessment, the results of LTNet are compared with the best method so far, $\text{SNet}_{SSL}^{(256)}$ for six scenes in Fig. 7.32. The scenes were chosen so that some show the differences in detail and others as an overall view. At a first glance, the predictions look relatively similar. Recalling the results from Figure 7.31, it can be seen that LTNet achieved generally a higher IoU for the classes *person*, *wall* and *traffic light*.

**Row 1 and 2** show examples where LTNet was able to successfully detect people, unlike SNet. Row 2 shows a scene where a person is standing in the entrance of a small shop. Unlike LTNet, $\text{SNet}_{SSL}^{(256)}$ was not able to segment the person in the entrance correctly. The pole in the foreground is also not as well segmented by SNet as by LTNet. On the other hand, LTNet classifies the bicycle trailer as a *person* and also suffers more from label-bleeding (seen in the red speckles around the trailer and the person). Label bleeding especially around pedestrians occurs very frequently by LTNet in the results of LTNet shown in Fig. 7.32.

**Row 3** is a representative scene showing almost all cases where LTNet performs better as SNet. First, LTNet successfully detected the people on the sidewalk, while SNet failed to do so. Secondly, the wall (blue-grey) and the fence (light-brown) along the pavement were also not detected by SNet, but almost completely detected by LTNet. Thirdly, the traffic signs (yellow) and traffic lights (orange) were slightly better detected by LTNet. Finally, on the right, one can see some poles that were partially classified as *vegetation* by SNet but were correctly detected by LTNet.

**Row 4 and 5** show two scenes from the same scanstrip. LTNet was able to recognize the people better in almost all cases. However, LTNet also suffers more from label bleeding and gives more inhomogeneous results than SNet. This can be seen, for example, on the facades in row 4, where many points were classified as *person*. A likely reason for this is that LTNet, unlike SNet, makes a point-by-point prediction. This is a problem that needs to be addressed in further research. A possible solution could be to adapt LTNet to make predictions for all 3D points within the input scanstrip patch.

### 7.6.4 Conclusion and Discussion

In this section LTNet from Chapter 4.3.2 was evaluated. In various ablation studies, LTNet has shown to outperform the previously presented methods for almost all classes. However, the use of self-attention in the multi-view subnet gave generally worse results, especially for smaller batch sizes. The possible reason for this is that after applying self-attention, the encoded multi-view observations are combined into a fixed-size vector, potentially rendering the relation of all observations useless. In a second ablation study, both sub-networks were tested and it was confirmed that LTNet can only achieve this quality by combining 2D and 3D observations. By comparing the confusion matrices created with and without the use of 3D features, it was shown that very similar patterns to naive label transfer emerge when no 3D features are used (Fig. 7.3). Finally, in the qualitative evaluation, LTNet was shown to be able to produce high quality semantic labels. However, as discussed earlier, this approach suffers from label bleeding and inhomogeneous predictions in some cases, which leaves room for improvement and further research

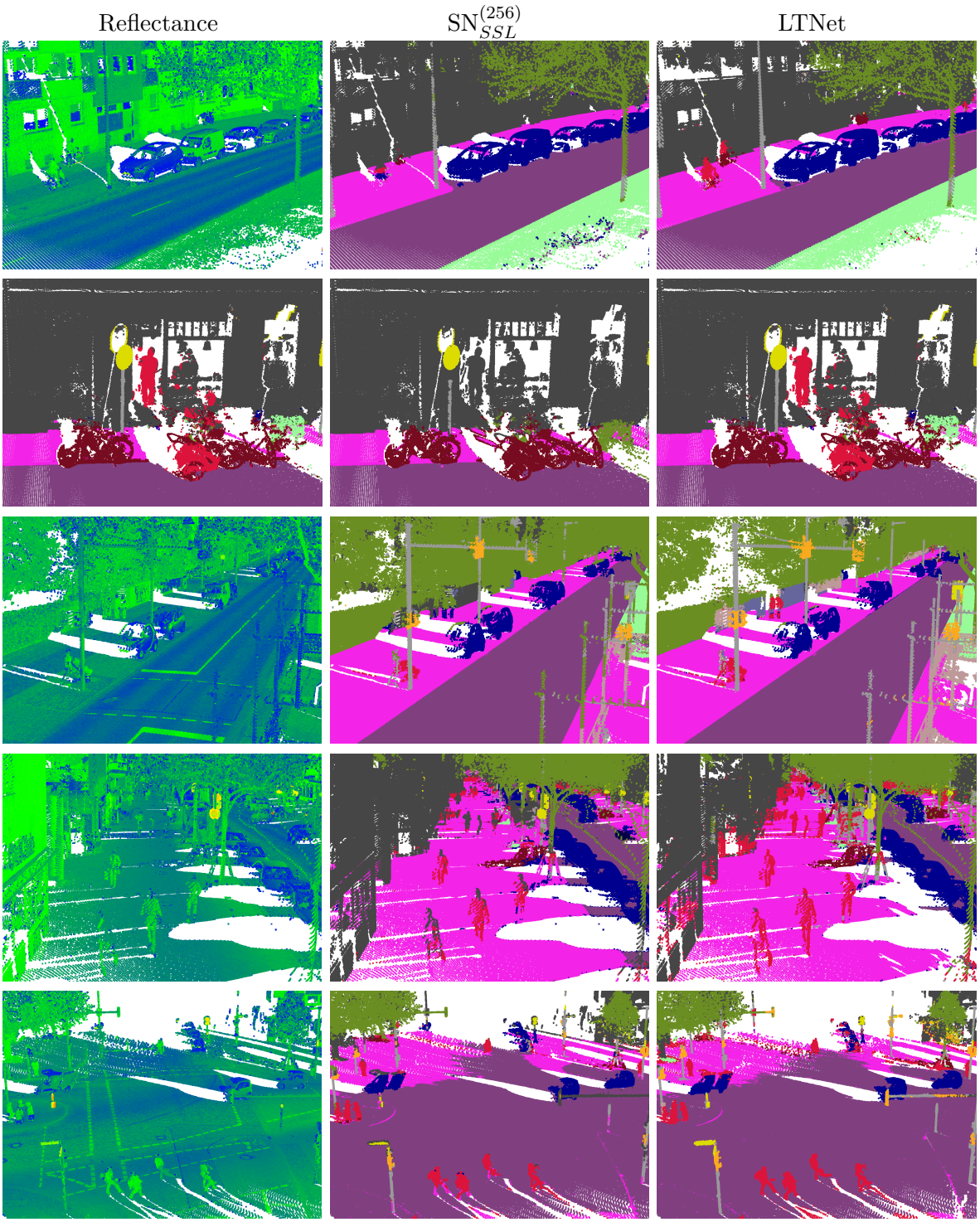



*Figure 7.32: Qualitative comparison of the semi-supervised scanstrip network $SN_{SSL}$ presented in Chapter 4.2.2 with LTnet for 5 different scenes (rows). The left column shows the reflectance values, the middle column the predictions by $SN_{SSL}$ and the right column the predictions from LTNet. In total, four different scanstrips from the test set are shown, with rows two and three as well as five and six belonging to the same scanstrip.*

## 7.7 Summary and Conclusion

This chapter systematically analysed problems and demonstrated solutions for label transfer from 2D to 3D data in a fully calibrated mobile mapping system. Using two reference sets, one in 2D and one in 3D, it was first analysed how good labels can be naively assigned to 3D points using majority voting over multiple 2D image pixel predictions. Figure 7.2 shows the influence of the domain gap between the Cityscapes and the MMS image data, as well as the amount of label errors in the image datasets and after label transfer in the 3D point cloud. The two confusion matrices (Fig. 7.3) revealed different error types, which are shown in the Table 7.1. In the subsequent sections of this chapter, this knowledge was used as a starting point.

In the second part, two "indirect" solutions were presented: The first solution used a gradient boosted decision tree trained on the 3D reference set for a pointwise label-noise correction. Even though this solution was the worst of all the trained models, it helped to gain insights into the features introduced. By using scanstrips, it was possible to apply classical 2D convolutional neural networks to the 3D data that are able to take the context of each 3D point into account. The introduced scanstrip network was able to beat state-of-the-art networks as well as two similar networks in the task of 3D semantic segmentation. Finally by additionally using unlabelled data the network could be trained in a semi-supervised fashion, which achieved even better results.

In Section 7.5 the Multi-View Networks were analysed that helped to bridge the domain gap between Cityscapes and MMS image datasets. Various ablation studies demonstrated the significant improvements achieved by Multi-View Networks as opposed to simple single-image networks. A possible reason for that could be that multi-view images help to increase the generalisation ability of the network by linking the reference data with other multi-view observations. Through the mechanism of self-attention, the MVNet also gained the ability to propagate information through all multi-view images to learn how to correct errors in the predictions. By using the predictions by MVNet as pseudo-labels to fine-tune a pretrained Deeplabv3+, it was shown that the domain gap can be reduced to a level very close to the original model with only 3-4 frames of training data.

In Section 7.6, the previously presented models were combined to create LTNet, a network that is able to learn the label transfer "directly" in an end-to-end manner. Consequently, this model achieved the best performance among all the competitors because it is able to parse multi-view image information and directly learns how to map the predictions into 3D space. Various ablation studies have shown the importance of all network parts and design choices.

# 8 Experiments and Results for Self-Supervised Completion

In this section, the methods evaluated in Chapter 7.1 are combined with self-supervised learning. First, the conditional GAN from Section 5.1 is tested. It learns to map from projected 3D point clouds to realistic-looking 2D RGB images. By applying a pretrained DCNNs to the synthesized images, the CGAN can act as an interface to map the 2D image predictions into a 3D point cloud without requiring access to a fully calibrated mapping system or even a camera.

In Section 8.2 the GAN from Section 5.2 will be tested. It is trained in a self-supervised manner to complete self-occluded 3D objects in voxel grids. Self-occlusion can lead to label transfer problems, resulting in label bleeding or blending between object edges. It is shown that the GAN is capable of completing different types of distinct object classes from only incomplete observations. By causing the discriminator to discard points with insufficient point density, the generator is forced to produce complete point clouds with high point density. Finally, it is shown that the GAN is capable of completing different types of synthetic data as well as from real car scans automatically collected from the MMS dataset and from KITTI.

## 8.1 Photorealistic Point Cloud Rendering

Photorealistic point cloud rendering requires the training of the CGAN presented in Section 5.1 on pairs of projected point clouds and RGB images. An additional input to the generator is the date of the captured image, which is mapped from a one-hot encoded vector onto the bottleneck of the generator. Using the month of acquisition as a parameter, it is shown that it is possible to map the same point cloud to many different possible outcomes, each resembling a different season. It is shown that the CGAN is capable of capturing different seasonal characteristics, such as snow in winter or green trees in summer, which it can predict for the same input point cloud. The prediction quality and generalization ability are evaluated using various metrics such as Multi Scale Structural Similarity (MS-SSIM) or Fréchet Inception Distance (FID). By semantically segmenting 2D real images and computing the mIoU to the corresponding classified generated 2D image, it can be shown that the synthesized images are realistic enough to be correctly interpreted by a pretrained DCNN. As the GAN can make predictions for almost any perspective, it will be shown that it is possible to combine different predictions to generate HD images. Finally the synthesized images will replace the MMS-images in the naive label transfer. It will be shown that this method suffers less from dynamic occlusions, even though no ground truth data was used.

### 8.1.1 Training Procedure

The CGAN was trained for 20 epochs with a batch size of one using the data presented in Section 6.2 using the hadoop rendering algorithm. As the CGAN only allows an input size of $512 \times 512$ pixels every input output pair was centrally cropped to a size of $1024 \times 1024$ pixels and then resized to $512 \times 512$ pixels using bilinear interpolation. Please note that only the central area has been cropped, as these areas are denser in the projected images than on the edges. For training the loss functions defined in Section 5.1 was optimized using Adam optimizer with a learning rate of $\lambda = 0.0002$ and $\beta_1 = 0.5$ for the discriminator and the generator.

### 8.1.2 Quantitative Evaluation

Evaluating a GAN is often not as straightforward as evaluating a classifier, s. (Shmelkov et al., 2018) or (Heusel et al., 2017). For quantitative evaluation, the Fréchet Inception Distance (FID) by Heusel et al. (2017) and the Multi Scale Structural Similarity (MS-SSIM) by Wang et al. (2004) and Wang et al. (2003) between the synthesized and the target images were calculated. The FID measures the Fréchet distance between the embedding distributions of the real and synthesized images. Both distributions are generated by extracting the features from the penultimate layer of a pretrained inception-v3 network. The synthesized distribution $X_g = \mathcal{N}(\mu_g,\Sigma_g)$ and the target distributions $X_t = \mathcal{N}(\mu_t,\Sigma_t)$ are modeled as multidimensional Gaussian distributions parameterized by their mean $\mu_t,\mu_g$ and covariance $\Sigma_t,\Sigma_g$. The FID score is calculated using the following equation:

$$FID = ||\mu_t - \mu_g||^2 + Tr(\Sigma_t + \Sigma_g - 2(\Sigma_t\Sigma_g)^{1/2}) \tag{8.1}$$

An FID score of zero corresponds to a perfect match between both distributions. The FID treats each image as a high-dimensional sample of a distribution that the generator must approximate. Multi Scale Structural Similarity, on the other hand, extracts metrics directly from the images and compares them. This involves calculating the distances between the luminance, contrast, and texture of the images at different scales (Wang et al., 2004). The score is then computed by the weighted product of all three terms. Both scores are calculated once for all training images to measure the overall performance and once per measurement campaign to see if the generator is able to capture the different seasonal characteristics of each campaign. In both cases, the images are generated in the same pose as the camera and according to the date of capture.

*Table 8.1: FID and MS-SSIM scores computed for every campaign. The closer FID is to zero the better. MS-SSIM is bound between -1 and 1, where 1 indicates a perfect match between real and generated images.*

| Campaign | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| FID ↓ | 13.1 | 15.5 | 12.9 | 12.4 | 11.9 | 14.3 | 14.8 |
| MS-SSIM ↑ | 0.54 | 0.47 | 0.54 | 0.59 | 0.56 | 0.5 | 0.55 |
| Campaign | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| FID ↓ | 15.4 | 13.3 | 12.7 | 12.7 | 14.0 | 31.0 | 34.2 |
| MS-SSIM ↑ | 0.51 | 0.55 | 0.54 | 0.54 | 0.49 | 0.43 | 0.43 |
| Campaign | | | | All | | | |
| FID ↓ | | | | 9.6 | | | |
| MS-SSIM ↑ | | | | 0.51 | | | |

Table 8.1 shows how well the CGAN is able to capture the characteristics of the individual mapping campaigns (Campaign 0-13) and the entire dataset (All). It was calculated on the data presented in Section 6.2. The results for FID and MS-SSIM show that the quality of the generated images is quite similar in all campaigns, except in campaigns 12 and 13. In campaign 12 and 13 the level of agreement between the generated and the real images is lower. A possible explanation could be that these campaigns took place in winter and were partly captured at night. As a result, many dark images with image noise were captured. In contrast, all other campaigns contain images takes in daylight. Therefore it could be assumed that the CGAN prefers to produce brighter images. To tackle this problem, the brightness of the images or time of the day could be used as parameters too, so that the CGAN can learn whether the target image should be bright or not. For comparison Atienza (2019) rendered point clouds from ShapeNet data. As a rough guide for the values in Table 8.1 and to show how well the CGAN performs, Atienza (2019) calculated a FID score of 31.5 and

a best-case MS-SSIM score of 0.64. The FID score by Atienza (2019) was also calculated using an inception-v3 network pretrained on ImageNet, which should make the two values comparable. However, it should be noted that both scores are difficult to compare because the methods were trained on different datasets. The FID score is also dependent on the number of samples, which should not be a problem in this case, because every campaign has roughly the same number of images.

The next experiment aims to provide information on how close the CGAN is to the real image according to a DCNN trained only on real images. For this purpose, the real images and the corresponding synthetically generated images were semantically segmented using Deeplabv3+. The semantically segmented real images are used as reference set and are compared with the synthetically segmented images. By measuring the mIoU between the two sets, it becomes clear how close the two sets are, because the DCNN should predict the same classes per pixel if the images are the same or at least very close. For this experiment two pretrained Deeplabv3+ models were used. One was trained on Cityscapes and the other one on PASCAL VOC 2012 (Cordts et al., 2016; Everingham et al., 2010). In contrast to Cityscapes the PASCAL VOC 2012 dataset has a class called background which can be useful in some tests. As test set a new and independent dataset was created, taken in February 2017 in Karlsruhe. This city was never part of the training set and will therefore be a good indicator whether the CGAN is able to synthesize RGB images for a unseen location. Apart from that, the Data was acquired with the same mobile mapping system and processed in the same way as presented in Section 6.2. Additionally, the CGAN is tested against the reference set for the MMS images presented in 6.1.1. Here, the GAN generates RGB images for each ground truth image, which are then passed to Deeplabv3+ (trained on Cityscapes). The predictions can then be compared to the reference set. However, due to dynamic occlusions, it is not possible to evaluate dynamic objects in any experiment. Table 8.2 therefore contains only IoU for static classes.

*Table 8.2: IoU between real and synthesized images (Karlsruhe) and also between synthesized images and the reference set 6.1.1 (Reference Set). As Baseline the IoU is also given between the original MMS-images and the reference set (MMS-Set). Predictions were made with Deeplabv3+ pretrained on Cityscapes*

| classes | Karlsruhe [IoU] | Reference Set [IoU] | MMS-Set [IoU] |
|---|---|---|---|
| road | 0.845 | 0.797 | 0.838 |
| sidewalk | 0.289 | 0.152 | 0.335 |
| building | 0.561 | 0.695 | 0.900 |
| wall | 0.186 | 0.013 | 0.215 |
| fence | 0.218 | 0.056 | 0.590 |
| pole | 0.177 | 0.144 | 0.700 |
| traffic light | 0.013 | 0.027 | 0.592 |
| traffic sign | 0.150 | 0.255 | 0.472 |
| vegetation | 0.806 | 0.687 | 0.870 |
| terrain | 0.330 | 0.208 | 0.716 |
| sky | 0.813 | 0.818 | 0.935 |
| mIoU | 0.399 | 0.350 | 0.651 |

Three columns are given in the Table 8.2. In the first two columns, the CGAN predicted RGB images, which were then semantically segmented by Deeplabv3+. For the first column, the IoU was calculated between the predictions by Deeplabv3+ on the original (real) Karlsruhe images and the synthesized ones. In the second column, the IoU was calculated between the predictions for the synthetically generated images and the corresponding human annotated reference set shown in Figure 6.3. For reference the third column shows the baseline prediction of Deeplabv3+ on the

real images with respect to the same reference set as in column two. These results have already been shown in Chapter 7.2.

The Table shows that Deepblabv3+ can recognize some of the synthesized classes quite well and others, such as. *traffic light*, barely at all. It should be emphasized that Deepblabv3+ suffers from a domain gap, as described in 7.2, which increases the error rate. For comparison: (Wang et al., 2018a) achieved an mIoU of 0.639 using a very similar approach. They semantically segmented synthetically generated Cityscapes images and compared them to the test set in Cityscapes. Apart from this, it can be seen that predictions on objects with a larger physical size such as roads, buildings, vegetation and sky achieve a higher IoU. The only exception here is the sidewalk; it is already known that Deepblabv3+ performs poorly on the MMS dataset (column MMS-Set row sidewalk), which could be the reason why it cannot detect this class in the synthesized images either. Finally, other errors such as calibration errors or occlusions may also affect the result. Regarding the "season" parameter, it should be noted that the CGAN has been set to generate images for the same date in which they were acquired in the mapping campaign. Overall the mIoU decreased by roughly 0.3 from the real MMS-images to the synthesized images. Later experiments will show that this can be compensated by using the procedure presented in Chapter 4.3

### 8.1.3 Qualitative Evaluation



*Figure 8.1: Input image (reflectance, left), synthesized image (middle) and real image (right)*

For the qualitative evaluation various Figures are presented. The first two show the general quality of the prediction and the ability to predict images corresponding to different seasons, s. Fig. 8.1 and 8.2. Figure 8.1 compares the input and the corresponding real image with the image generated by the CGAN. The predictions are relatively close to the original, but the colors are different in some cases. For example, in the real image there is a turquoise van in the background, which is silver in the generated image. Even more interesting is that the generated house has white walls and a red roof, which is very common in this area. The real image, on the other hand, shows red walls and a black roof. This suggests that the color information could be derived mainly from the shape of the object. However, later experiments will show that the reflectance value also encodes textures.



*Figure 8.2: Summer (middle) and winter (right) representation of the same input point cloud (left)*

The image pairs in Figure 8.2 show images predicted for different seasons side by side. One image shows the result for a month in summer and the other for winter, both with typical-looking features. The summer image more leaves on the trees and a different colour scheme, while the winter image has fewer leaves on the trees, snow on the roads, and more wet-looking roads. But the trees in the summer image have relatively few leaves. A possible reason for this could be that the point cloud was recorded in winter, which means that the trees have fewer leaves in the input, making it harder to map from winter to summer. However, later experiments show that in addition to the parameterized month, the reflectance values of the input encode mostly how full a tree looks.




*Figure 8.3: Randomly sampled pairs of predictions (left) and ground truth images (right) for every campaign. The pairs are sorted by campaign number (campaign 0 is top left and 13 bottom right).*

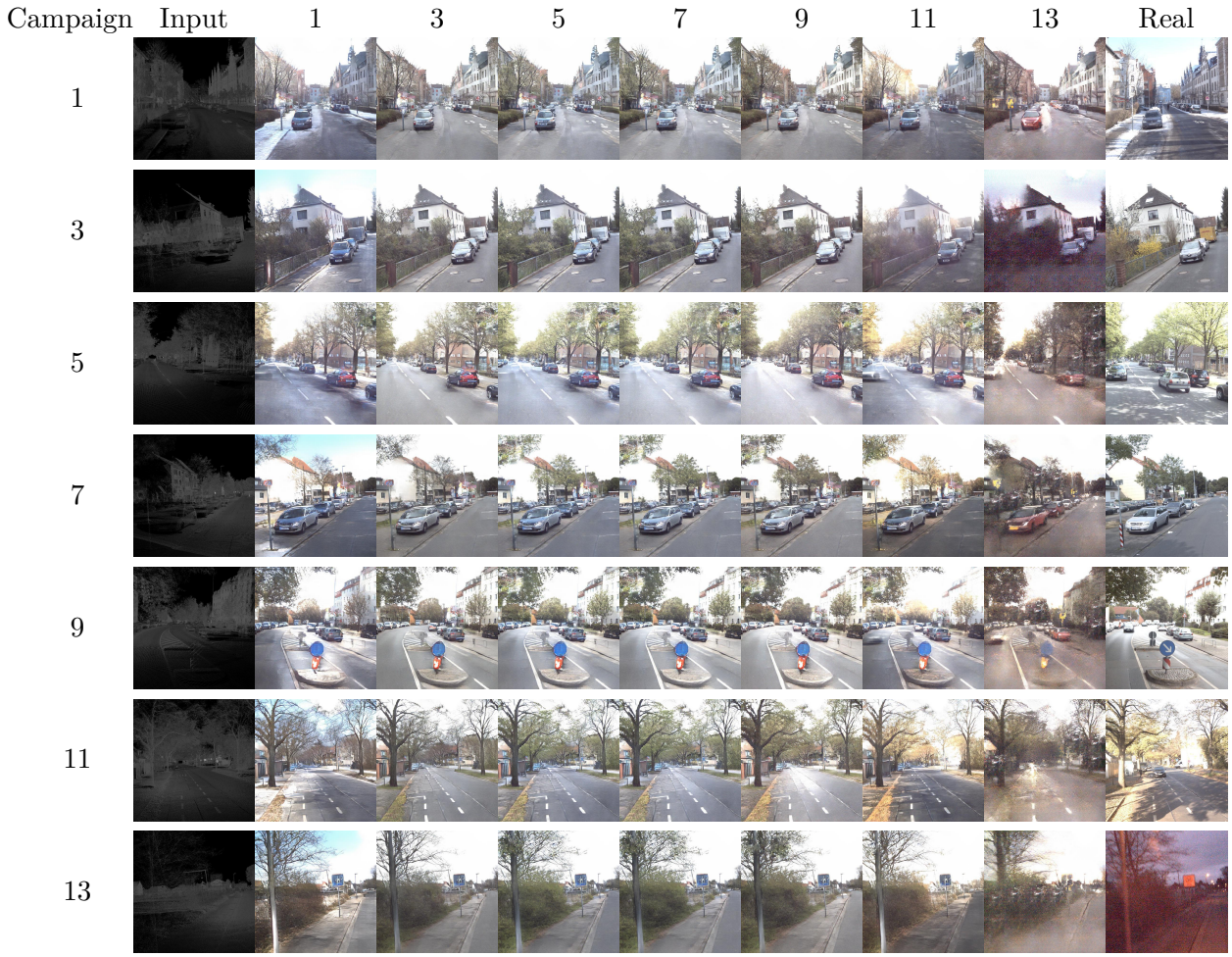
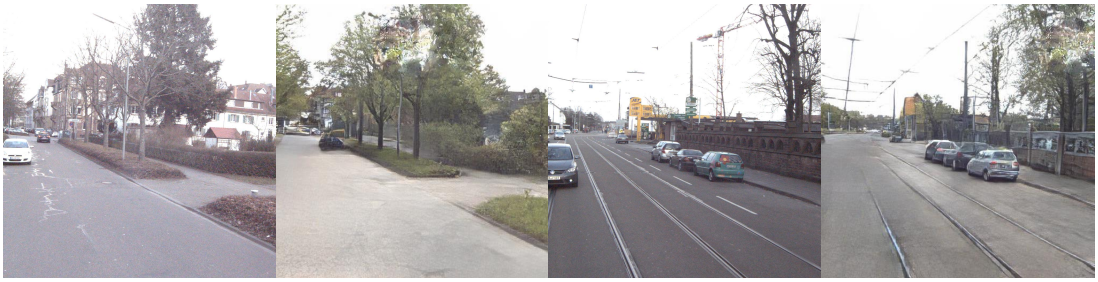



*Figure 8.4: Different representations for the same input point clouds. Each row shows an example from one campaign. The columns show the input (left), the different predicted Campaign (1-13) and the corresponding real image (right).*

Figure 8.3 and Figure 8.4 show an overview of the mapping campaigns and their different characteristics. Figure 8.3 shows randomly selected pairs of synthesised and real images for each mapping campaign (14 pairs in total). The mapping campaigns were taken over a whole year and all show different seasonal features, colors and times of day. One can see that the network produces images that are very similar to the real image: The last campaign, for example, contains dark images with a lot of image noise, which is also present in the generated image by the CGAN. It is noticeable that when the camera has lens flares, they do not appear in the predictions.

The grid in Figure 8.4, on the other hand, shows randomly selected input images mapped to the style of other campaigns to show the different possible outcomes for the same input. At first glance, it is clear that each column looks roughly the same in terms of color, time of day and season. For example, the second column (campaign 1) shows sparse tree canopies and snow on the roads regardless of the campaign. The following columns show that the color changes depending on the seasonal characteristics. Finally, the last predicted column shows a campaign taken mainly at dawn. Here the predicted images show dark colours and the typical image noise patterns due to low light.



*(a)* Winter (Real) *(b)* Summer (Synthetic) *(c)* Winter (Real) *(d)* Summer (Synthetic)

*Figure 8.5: Image pairs of Karlsruhe in winter and the corresponding synthesized summer images.*



*Figure 8.6: Cars successfully classified in synthetic images (bright grey) using Deepblabv3+ pretrained on PASCAL VOC.*

To qualitatively show the predictive and generalisation capabilities of the CGAN, the figures in 8.5 and 8.6 present some results for the Karlsruhe dataset. Figure 8.5 shows images of two scenes from Karlsruhe, both taken in February. The predictions by the CGAN are made for a month in summer to increase the difficulty. The results show that the CGAN predicts realistic looking images that are very close to the original ones. Obviously, dynamic objects do not appear in the same place due to dynamic occlusions. To show that the CGAN can also generate realistic cars, the images in Fig. 8.6 (first row) were semantically segmented using a pretrained Deepblabv3+ Fig. 8.6 (second row). In almost all cases, the cars are correctly detected, showing the ability of the CGAN to predict realistic looking cars.

Finally in order to create images, that have the same size as the original MMS-Images a method for image stitching was presented in Algorithm 5 in Section 5.1.3. The result of this algorithm can be seen in Figure 8.7. The image on the left shows the centrally cropped prediction. The images in the middle and on the right shows the stitched high resolution images, both using different weight matrices $W$. The choice of $W$ controls how the predictions merge into each other. If $W$ is filled with ones, the mean value of the RGB values in the overlapping windows is calculated, resulting in "block-like" artifacts, see Fig. 8.7 middle. The predictions will blend more smoothly if a Gaussian



*(a)* Single prediction *(b)* Mean stitching *(c)* Smooth stitching

*Figure 8.7: The example shows two ways of stitching different synthetic images into one large image. The left image shows a single synthetic image with* $512 \times 512$ *pixels. The middle and right images show the same scene, but with a resolution of* $2056 \times 2452$*, created by stitching many synthetic images together. Image b was created by calculating the average RGB value of the overlapping windows, and for image c the overlapping windows were weighted with a Gaussian mask to achieve a smoother result.*

weighting mask $W(n,m) = \frac{1}{2\pi\sigma^2} e^{-\frac{n^2+m^2}{2\cdot\sigma^2}}$ with its peak in the center of the matrix is used, see Fig. 8.7 right image.

### 8.1.4 Multi-View Error Correction in GAN Images

The experiments in Section 7.5.7 have shown that multi-view outlier corrections can be used to close the domain gap for a pretrained DCNN. By combining this method with the CGAN it is possible to achieve reasonable prediction quality on the synthesised GAN images to detect most of the static objects and transfer labels into 3D. For this purpose, the stitched synthetic images of the CGAN are passed to Deeplabv3+ and the MVNet is trained on the semantically segmented images to improve the predictions Deeplabv3+. Predictions made with Deeplabv3+ on the stitched synthetic images reach an mIoU of 0.41 on the reference set shown in Fig. 6.3, which needs to be improved in this section.

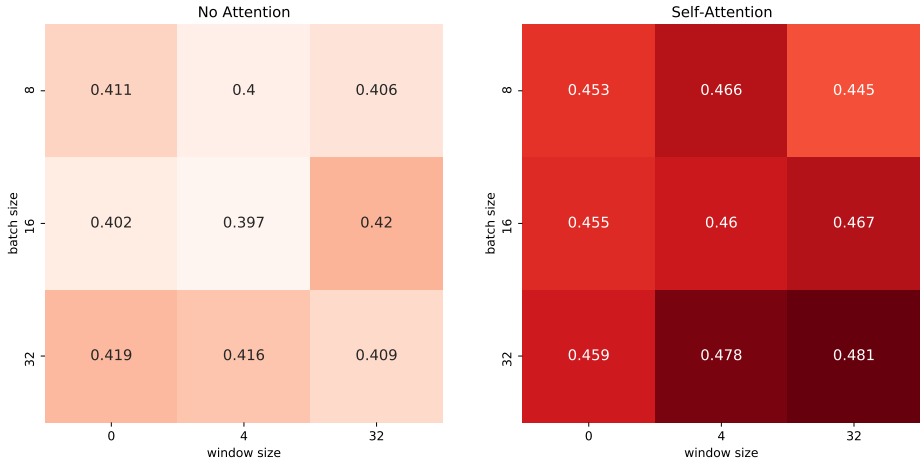



*Figure 8.8: Resulting mIoU for ablation studies conducted with and without self-attention on the semantically segmented synthesized CGAN images. The average mIoU using no attention is 0.407 and with self-attention the average mIoU is 0.457.*

Using exactly the same training, validation and test sets, almost the same hyperparameters and the training strategy presented in Section 7.5, the multi-view network was trained on predictions made by Deeplabv3+ on the synthetic images. First, a grid search was performed to find the optimal batch size $b \in \{8,16,32\}$ and window size $w \in \{0,4,32\}$. Instead of Dice Loss, the network was trained with cross entropy, which gave slightly better results. Apart from this, nothing was changed from the original training procedure. Figure 8.8 shows the performance on the test set using self-attention. The results show that the use of self-attention outperforms the simple single-view networks, achieving a maximum mIoU of **0.481** on the test set.

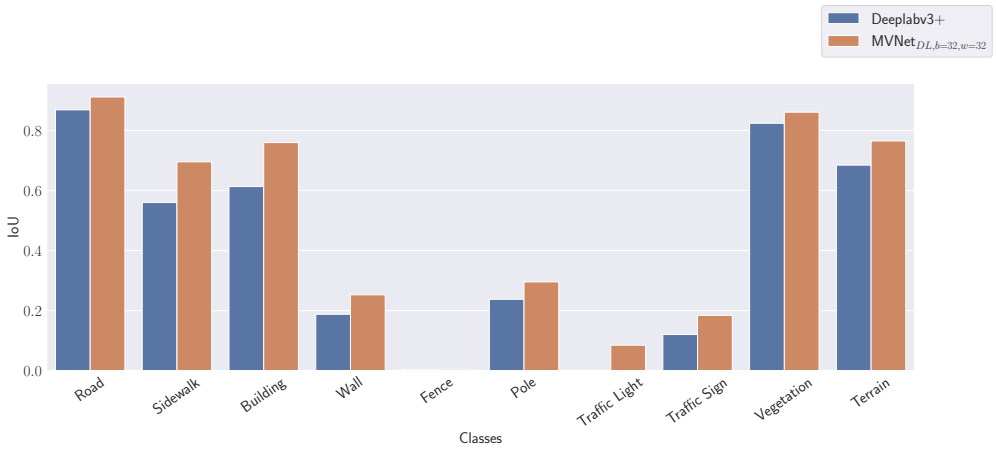



*Figure 8.9: Comparison of the IoU values for predictions with Deeplabv3+ on the fake images and the same predictions corrected by MVNet trained with parameters $w = 32, b = 32$. The mIoU for Deeplabv3+ is 0.41 and the mIoU of MVNet is 0.481*

A look at Figure 8.9 shows that all classes have improved compared to the original predictions of Deeplabv3+. The mIoU of Deeplabv3+ before correction is **0.41**. Comparing this result with the left-hand side of Figure 8.8 shows that the network without attention did not increase the mIoU at all. The average mIoU for all ablation trials without attention is 0.407, which means that using no attention leads to worse results on average than the naive (uncorrected) baseline. In the following, the best Multi-View Network that has achieved the highest mIoU is used for all subsequent evaluation and fine-tuning steps.

#### 8.1.4.1 Finetuning Deeplabv3+

The procedure for fine-tuning Deeplabv3+ on the CGAN images is exactly the same as in Section 7.5. In that section, a training set of 2636 image pairs of MMS-RGB and label images was created by correcting the original Deeplabv3+ predictions using MVNet. Here, the CGAN is used instead to generate MMS-RGB images for exactly the same images as in Chapter 7.5. For each image, the pretrained Deeplabv3+ makes predictions that are corrected by MVNet. As MVNet performs better than Deeplabv3+ in each class, all predictions for static classes are kept by MVNet. Only the predictions for class *sky* and dynamic classes are retained in label images. Deeplabv3+ is then fine-tuned using the same hyperparameters as in Chapter 7.5. Since this step has already been extensively tested on the MMS images, no further ablation studies are performed in this step. Please note that due to hardware limitations the fine-tuning step is again done for Deeplabv3+ with xception-65 backbone, in contrast to the network presented in the baseline using xception-71.

Figure 8.10 shows examples of the dataset before and after correction. The left side shows the synthesised RGB images. The second column shows the original (uncorrected) predictions from

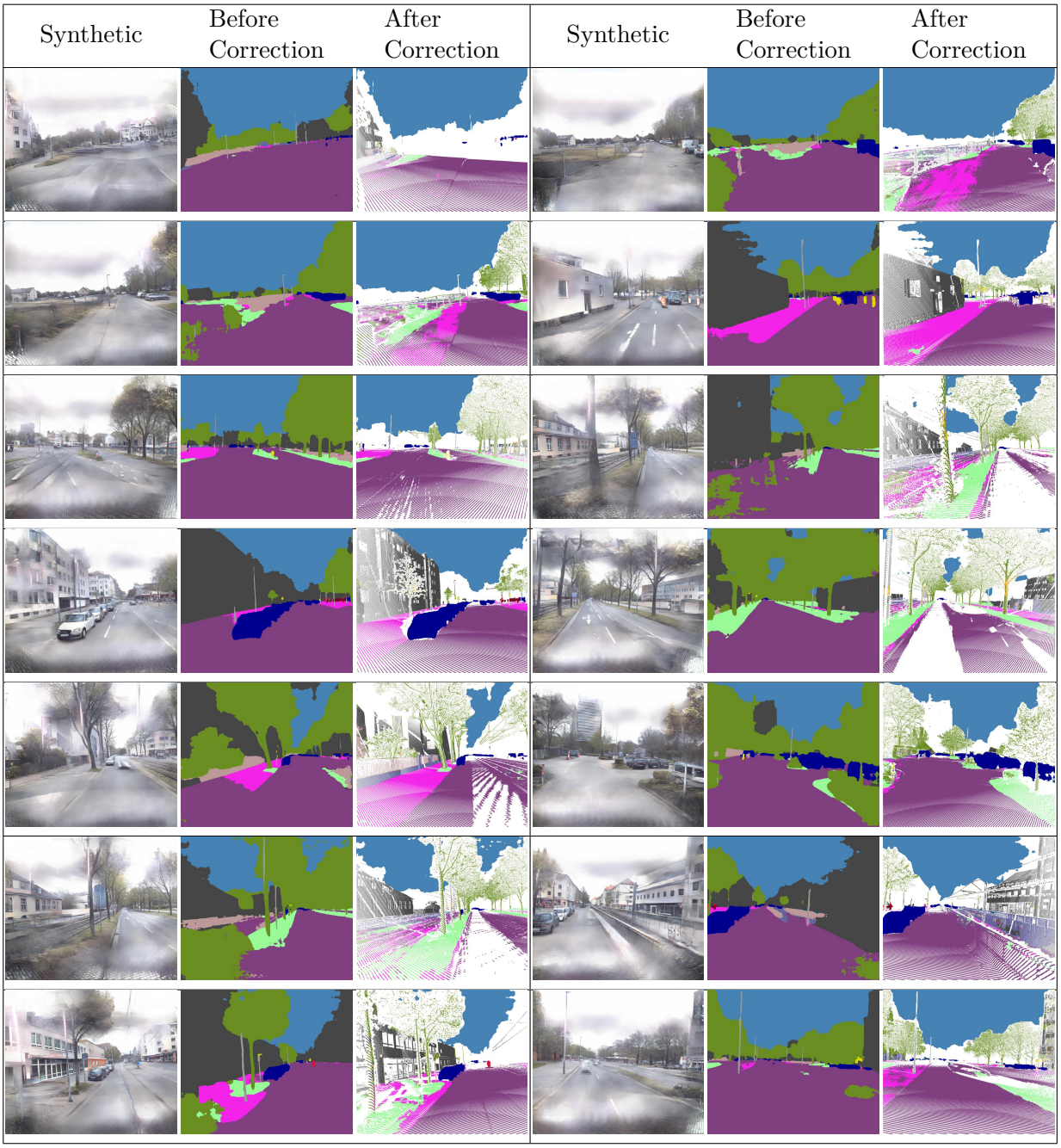



*Figure 8.10: Examples showing randomly sampled Deeplabv3+ predictions on synthesized stitched RGB images (Synthetic) before correction and after correction with MVNet.*

Deeplabv3+. The third column shows the corrected predictions, which serve as labels in the fine-tuning step. Comparing the results before and after correction, one can see that MVNet has recovered many classes that were not present before. This is particularly clear for the classes *traffic light* and *traffic sign*, which were not present in most of the original predictions. The prediction for *sidewalk* and *building* has also been qualitatively improved. For example, buildings are less likely to be confused with trees and the typical problems with the *sidewalk* class have been alleviated.

#### 8.1.4.2 Transfer Classified CGAN Images to 3D

In this section, label transfer from synthesized RGB images to 3D point clouds is demonstrated. This is done to see if it is possible to perform a naive label transfer without the use of a mobile mapping system, using only synthesized RGB images as an interface, so that predictions for synthesized images can be mapped onto 3D point clouds. As a reminder, in Chapter 7.2, the Deeplabv3+ (Xception-71) predictions on MMS images were mapped to 3D point clouds. Each 3D point was assigned to the majority label among all corresponding 2D image pixel predictions. The quality of the result was then measured by computing the IoU between the mapped labels and the human annotated reference set presented in Chapter 6. To make the following experiment comparable, nothing is changed except that the MMS images are replaced by the synthesized CGAN images. The RGB images are generated for exactly the same camera positions as the real RGB images in the original baseline. The procedure is as follows:

1. The point cloud is projected into images. This is done for all scenes where annotated 3D point clouds are available.
2. The stitched RGB images are generated using Algorithm 5 for each projected image.
3. All synthesised images are semantically segmented using the either the pretrained Deeplabv3+ with Xception-65 or 71 backbone or the Deeplabv3+ that is finetuned on the CGAN images in Section 8.1.4.1.
4. The predictions are mapped back to the point cloud using the naive label transfer with majority voting.

This process leads to three different results: 1.) Predictions mapped from 2D to 3D based on the CGAN images from Deeplabv3+ with xception-65 backbone trained on Cityscapes, denoted as **X65(GAN)**. 2.) The predictions mapped from 2D to 3D of the same network which was additionally fine-tuned to the corrected labels, denoted as $\mathbf{X65(GAN)}_{fine}$ and 3.) the mapped predictions based on the CGAN images of the Deeplabv3+ network with xception-71 backbone used in the baseline in Section 7.2, denoted as**X71(GAN)**. All this is compared to the original baseline results obtained by using Deepblabv3+ with xception-71 backbone on the real MMS images shown in Figure 7.2 denoted as $\mathbf{Deeplabv3+}_{3D}$.

| Image Source | Real Images | Synthetic Images | | |
|---|---|---|---|---|
| Method | $\text{Deeplabv3+}_{3D}$ | X71(GAN) | X65(GAN) | $\text{X65(GAN)}_{fine}$ |
| mIoU | **0.481** | 0.35 | 0.329 | 0.384 |

*Table 8.3: Comparison of achieved mIoU for all label transfer experiments. $Deeplabv3+_{3D}$ is displayed as reference when real MMS images are used as source for the label transfer. The others are using synthetic images with different pretrained DCNNs for semantic segmentation.*

Figure 8.11 show the IoU per class for all different experiments. The results are very interesting because although Deeplabv3+ performs worse on the synthetic images than on the real images, the results show that in some cases the IoU in 3D is higher when using the synthesized images.

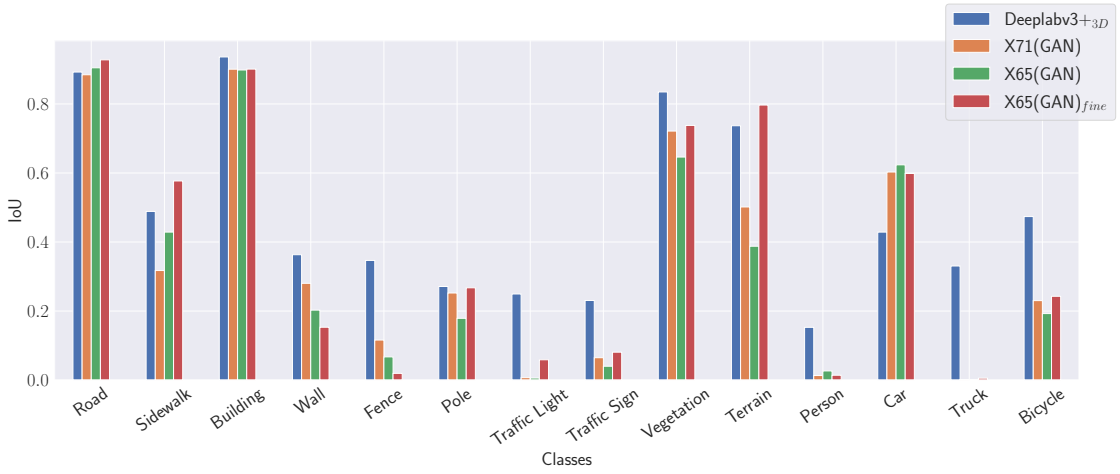



*Figure 8.11: Comparison of the results obtained when using naive label transfer from images to 3D for different image sources. Deeplabv3+$_{3D}$ is a reference that shows the results when real MMS-images were used as source, see Section 7.2. For synthetic images Deeplabv3+ with Xception-65 or 71 backbones is denoted by X65 or X71. When predictions are based on the synthesized images, this is denoted by (GAN). When Deeplabv3+ was fine-tuned with MVNet this is denoted by $fine$.*

The most prominent case here is the class of *car*. Cars suffer greatly from the problem of dynamic occlusion, since laser and camera beams often do not point to the same object when the object is moving. In Section 7.2 cars have been shown to be well detected in generated images. The Figure 8.11 shows that the CGAN tends to generate cars that match the point cloud, resulting in a significantly higher IoU. However, the CGAN still fails to generate people, trucks or bikes well enough to match the performance for cars. The reason for this could simply be that these classes are less frequent in the images or are smaller in physical size, so they are neglected by the CGAN. Nevertheless, it shows that this could be a promising direction for the treatment of dynamic occlusions and can be part of further research. Except for the class *car*, the synthesized images match the performance of the real images for many other classes. The results in Figure 8.11 show that classes with large physical size such as roads, buildings or vegetation are very similar to the IoU obtained when using real images.

Figure 8.12 shows exemplary results of label transfer with naive label transfer. The left column shows the scene colored by reflectance as a reference. The point clouds in the middle and right columns were both classified using the naive label transfer method. The difference in both results is only that real images were used for the middle column and the synthesized images for the right column. It should be clear to see that Deeplav3+$_{3D}$ suffers from dynamic occlusion, as shown by the blue labels on the road (first and second row, center). The X65(GAN)$_{fine}$, on the other hand, shows no signs of dynamic occlusion. The last row shows that the X65(GAN)$_{fine}$ detects parked cars well. The cars appear in the correct place in the generated images, so there are fewer errors when transferring the labels to 3D than when using real images as a source.

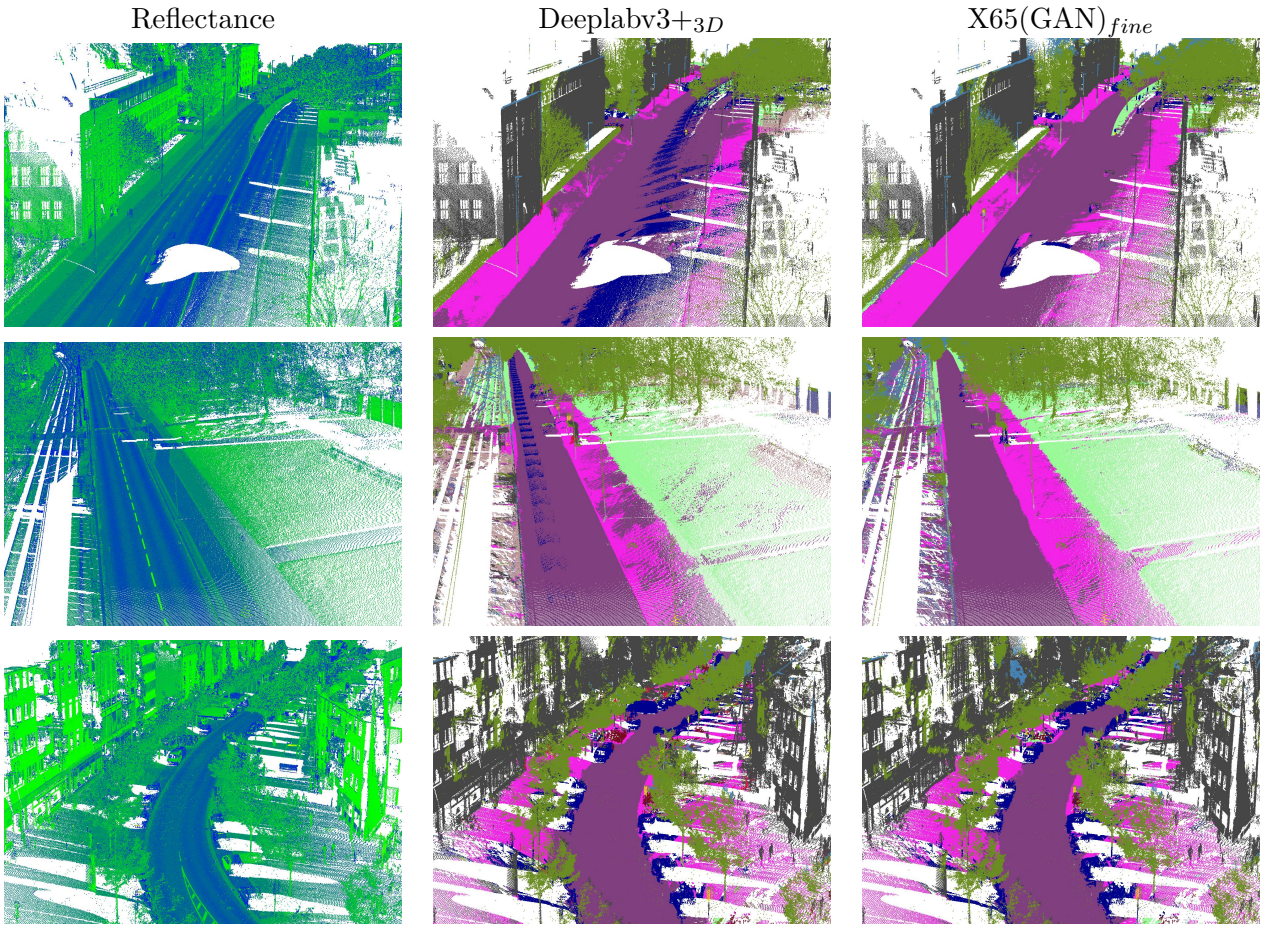



*Figure 8.12: Three example point clouds (left) showing the effect of naive label transfer using MMS-Images (middle) or synthesized GAN images (right)*

### 8.1.5 Conclusion and Discussion

In Section 8.1, the CGAN for photorealistic point cloud rendering presented in Chapter 5.1 was extensively tested. The GAN was trained on a subset of the MMS dataset containing over 250k pairs of projected point cloud images and real RGB images. The quality of the prediction was shown by first calculating the FID and MS-SSIM values for each measurement campaign and then for all data. This test showed that the GAN was able to detect seasonal features of each month in which a measurement campaigns was recorded. It also showed that the GAN is able to replicate similar lighting, time of day and colour features as in real camera images. To further test how well the CGAN generalises to new, previously unknown locations, a second dataset was created. Here, the similarity between the predictions of a pretrained Deeplabv3+ between the synthesised images and the real MMS images was measured. This experiment showed that static classes of objects with large physical size could be generated quite well, whereas other object classes corresponding to thin or smaller objects were not generated as well. By comparing the mIoU with the corresponding real image predictions on the same reference set, the performance of the GAN could be evaluated. After the quantitative evaluation, the CGAN was evaluated qualitatively. It was shown that the GAN images can be sequentially merged to produce an image of any resolution, producing very high resolution images. It was also shown that the CGAN was qualitatively able to predict cars that could be successfully detected and semantically segmented by Deeplabv3+.

In the final part of the experiments, the synthesised GAN images were used to map predictions of a pretrained DCNN from 2D images onto 3D point clouds without the need for a fully calibrated mobile mapping system or even a camera. In this process, the camera images were replaced by the synthesised GAN images, which serve as interface between the pretrained DCNN and the 3D point cloud. By training the MVNet on a very sparse reference set, equivalent to about 3-4 MMS images, to learn how to correct the DCNN predictions on the generated images, the quality of the label transfer was significantly improved. It was shown that naive label transfer using the synthesised images successfully removes labelling errors caused by dynamic occlusions in cars. This is particularly interesting as these results can also be obtained without any reference data (X65(GAN) and X71(GAN)).

further improvements are under discussion for this method. It would be interesting to increase the overall prediction quality by using more GPU memory, which would allow the use of a larger CGAN model. To further improve the FID and MS-SSIM to make the CGAN better reflect seasonal characteristics, the discriminator could be given the current season as an additional input, as in ACGAN by Odena et al. (2017), forcing the generator to take the season into account, as it would otherwise be exposed to the discriminator. Training of the MVNet on the synthesised images could be vastly improved by augmenting MVNet using the properties of the CGAN. First, one could generate images with all possible seasons for each point of view, increasing the training dataset by the amount of mapping campaigns. Second, as shown, the Multi-View Network benefits significantly from multiple views and predictions of the same object. Because the CGAN can generate any number of multi-view images with potentially very low calibration errors, the Multi-View Network can access any other view of the same object without having to record the data on site, simply by generating it. For example, trajectories could be created where the virtual camera orbits an object for which the CGAN generates images to create all possible views of the same object. Finally, the same property could be used to generate any number of training examples to train a 2D DCNN, provided the input point cloud is labelled.

## 8.2 Self-Supervised Shape Completion

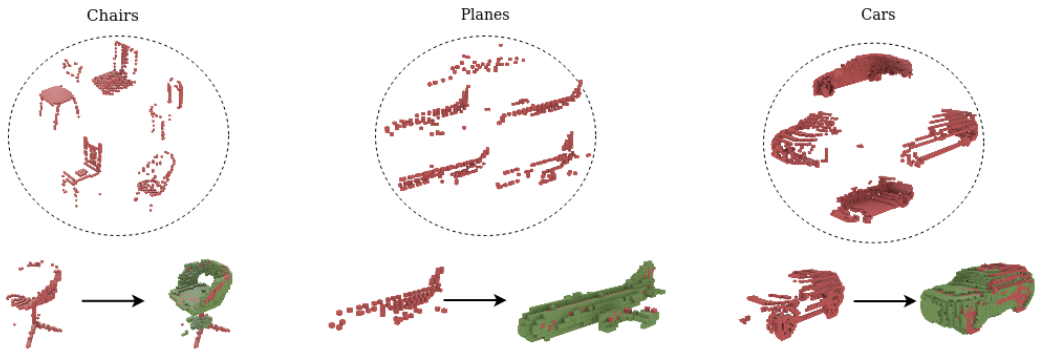



*Figure 8.13: The GAN learned to complete shapes (green voxels) from sparse observations only (red voxels)*

As mentioned in Section 5, self-occlusions are an inherent problem in label transfer. Occluded voxel regions have an unknown state, because they can be either occupied or free. Ray tracing can fail in these regions as the voxels have not been measured and are shown as unoccupied. If camera and laser beams do not match, this can result in label bleeding around these objects. Using learning based approaches to estimate these regions often requires large amounts of training data, which is difficult or even impossible to obtain in real urban environments due to the high variety of different object classes. In this section, the approach described in 5.2 is tested on the synthetic and real data presented in Section 6.3. It is shown qualitatively and quantitatively that the method is able to learn to estimate occupied voxel cells of the occluded part of an object class in a self-supervised fashion without any label data.

### 8.2.1 Training Procedure

For training, the complete architecture, as shown in Fig. 5.4, was implemented in Tensorflow and trained end-to-end. The GAN is trained on the dataset with incomplete cars from the MMS presented in Section 6.3. Additionally the GAN is trained and tested on the synthetic datasets presented in Section 6.3.2 for which a complete reference is available. Because the GAN is trained unsupervised, it is trained on all available data and tested on the complete reference shapes. For all results shown, one GAN was trained per class respectively per dataset. Training is run with batch size 5 for 5 epochs. The generator is trained with 50% drop-out, see Fig. 5.7, the discriminator without drop-out. As optimiser Adam is used with different learning rates of 0.0001 for the discriminator and 0.001 generator (a strategy sometimes referred to as TTUR, (Heusel et al., 2017)). During the training the same number of subregions is used in all experiments, so that each subregion has a size of $16^3$ voxels. This means that for a voxel grid of $32 \times 64 \times 32$ voxels, the subregion size is $n_x \times n_y \times n_z = 2 \times 4 \times 2$ regions. Empirically, the training is stable across a range of different subregion sizes, and also leads to similar results. The hyper-parameter $m$ that selects the complete subregions is set conservatively, in this implementation to 25% of all blocks.

### 8.2.2 Quantitative Evaluation

The evaluation is done on two types of datasets. First, there is a synthetic dataset for which ground truth is available because the occluded samples are created from watertight meshes that serve as ground truth (see Section 6.3.2). The second dataset uses real point cloud data for which ground truth is not available (see Section 6.3). It serves to show that the GAN can be applied to real data and that it generalizes to different laser scanners with different characteristics.



*Figure 8.14: Prior shapes that were used for the quantitative evaluation process. The pictures show from left to right: car, plane, chair and bathtub gathered from Shapenet and Modelnet.*

*Table 8.4: Quantitative Results for the proposed method, listed her as GAN. Numbers marked with † are taken from Stutz and Geiger (2018). Note that they use a slightly different voxel grid for the ShapeNet classes.*

| | | | ShapeNet Cars | | | ShapeNet Planes | | |
|---|---|---|---|---|---|---|---|---|
| Method | Complete supervision | Resolution | mIoU↑ | Acc↓ [vx] | Comp↓ [vx] | mIoU↑ | Acc↓ [vx] | Comp↓ [vx] |
| Dai et al. (2017) | 100% | 32×72×32 | 0.87† | 0.32† | 0.56† | – | – | – |
| Stutz and Geiger (2018) | ≤7.7% | | 0.78† | 0.54† | 0.74† | – | – | – |
| Dataset | 0% | 32×64×32 | 0.09 | – | 2.9 | 0.49 | – | 1.75 |
| Prior Shape | | | 0.64 | 1.07 | 0.96 | 0.36 | 1.49 | 1.49 |
| GAN | | | 0.70 | 0.78 | 0.59 | 0.54 | 0.25 | 0.65 |

| | | | ModelNet Bathtubs | | | ModelNet Chairs | | |
|---|---|---|---|---|---|---|---|---|
| Method | Complete supervision | Resolution | mIoU↑ | Acc↓ [vx] | Comp↓ [vx] | mIoU↑ | Acc↓ [vx] | Comp↓ [vx] |
| Dai et al. (2017) | 100% | 32×32×32 | 0.59† | – | – | 0.61† | 0.66† | 0.67† |
| Stutz and Geiger (2018) | ≤10% | | 0.50† | – | – | 0.41† | 1.49† | 1.07† |
| Dataset | 0% | | 0.19 | – | 2.75 | 0.21 | – | 1.85 |
| Prior Shape | | | 0.17 | 1.06 | 1.68 | 0.15 | 1.70 | 2.20 |
| GAN | | | 0.34 | 0.90 | 0.99 | 0.33 | 0.88 | 1.58 |

| | | | MMS Dataset | | | KITTI (trained on MMS) | | |
|---|---|---|---|---|---|---|---|---|
| Method | Complete supervision | Resolution | | Acc↓ [m] | Comp↓ [m] | | Acc↓ [m] | Comp↓ [m] |
| Prototype Car | 0% | 32×64×32 | – | – | 0.12 | – | – | 0.09 |
| GAN | | | – | – | 0.03 | – | – | 0.08 |

Please note for the synthetic datasets Stutz and Geiger (2018) predicted only filled shapes. To make the results comparable, all synthetic *car* predictions were post-processed, by flood-filling them using a graph cut algorithm, where the graph source and sink nodes are the barycenter and the boundary of the voxel volume, and an exemplar car (see Fig. 8.14 on the left) serves as prior to determine the edge weights. As the other shape categories are not as compact as cars, they were not post-processed.

The quality of shape completion is measured by calculating, for every category, the mean intersection over union (mIoU), accuracy (Acc) and completeness (Comp) of the predictions w.r.t. the ground truth shape. Accuracy is defined as the average distance from the predicted shape to the ground truth target. This is done by calculating the euclidean distance for every point of the prediction to the nearest point on the reference shape. Completeness is the average distance in the opposite direction, i.e., the distance from the target to the predicted shape. The results are compared to those obtained by Stutz and Geiger (2018) and Dai et al. (2017), both using (semi-) supervised approaches. For synthetic data, the accuracy and completion is reported in multiples of the voxel grid size [vx] and for KITTI and MMS in meters.

As a simple baselines for the unsupervised setting, a fixed template is also used as "completion result" instead of the generated output. The results are shown in Table 8.4 in the rows called "Prior Shape". For evaluation of Modelnet and Shapenet the first sample from the ground truth was picked as completion result. The images in 8.14 show the shape for every category. It should become clear that these shapes do not look unusual or particularly rare. For the MMS dataset and KITTI this "prior shape" is the prototype car, see Fig. 6.10c. Additionally the rows called "Dataset" in Table 8.4 show the effect of the occlusion. Here all generated incomplete samples were compared to the ground truth.

Table 8.4 shows the quantitative results obtained on the synthetic and real datasets. The first column shows the different methods. The second column (Complete supervision) gives an estimate of how much these methods rely on the use of reference data for training. The size of the voxel volume is given in the third column (Resolution). Results are not available for some methods because they were not presented in the work of Stutz and Geiger (2018) or because there is no reference data available for the MMS dataset and KITTI.

A look at Table 8.4 shows that the GAN increases the mIoU compared to the fixed prior shapes, which confirms that the generator does condition its predictions on the data. Please note that testing the shape completion outputs of the GAN against ground truth is somewhat problematic, for two reasons. First, there are many plausible completions of a partial shape, if a predicted shape does not match the ground truth that does not imply that it does not match another real instance of the target category. Second, for every soft prediction a threshold is applied to decide whether a cell is occupied or not. Differences to the ground truth can therefore also be due to aliasing and noise, in situations where the soft posterior is well estimated. For some applications, e.g. localisation or shape comparison, it is not necessary to threshold the probabilistic occupancy grid, but it is advantageous to work with soft occupancy, which is more forgiving when processing voxels. In the case of label transfer, the predicted probability that a cell is occupied could also be used for a "soft occlusion filter". This could be implemented by weighting the rays passing through soft grid cells by the predicted probability. So instead of increasing the label histogram by one, it would be increased by one minus the probability that a cell is occupied. However, the development and implementation is still the subject of further research.

### 8.2.3 Qualitative Evaluation

Figures 8.15 and 8.16 show example predictions of the GAN. Figure 8.15 shows the results for all synthetic datasets that had low point density. Figure 8.16a shows the results for the high density datasets and Figure 8.16b shows the result for the real datasets.

Overall, the results show that the generator is able to successfully complete shapes with a meaningful prediction. It produces a variety of subtypes that differ in size and shape. For example, Shapenets plane dataset contains different types of aircrafts, such as passenger planes, helicopters and even spaceships. It can be seen that the GAN responds appropriately to the input. While some predictions are certainly more plausible than others, there were no error cases observed where the shape was rendered completely unrecognisable. Please note that the results presented have not been cleaned or post-processed in any way.

In Figure 8.15 the results for the category *car*, *plane*, *chair* and *bathtub* with lower scan resolution are presented. As can be seen, the GAN is able to complete shapes even from very sparse data. The results show that it is able to fill most of the incomplete shapes. In particular, the category *chair* can be considered a hard problem due to the high class variability. The dataset contains chairs with different types and numbers of legs, as well as with different chair backs and armrests. It can be seen that the generator predicts the correct type of legs and generates armrests when it is

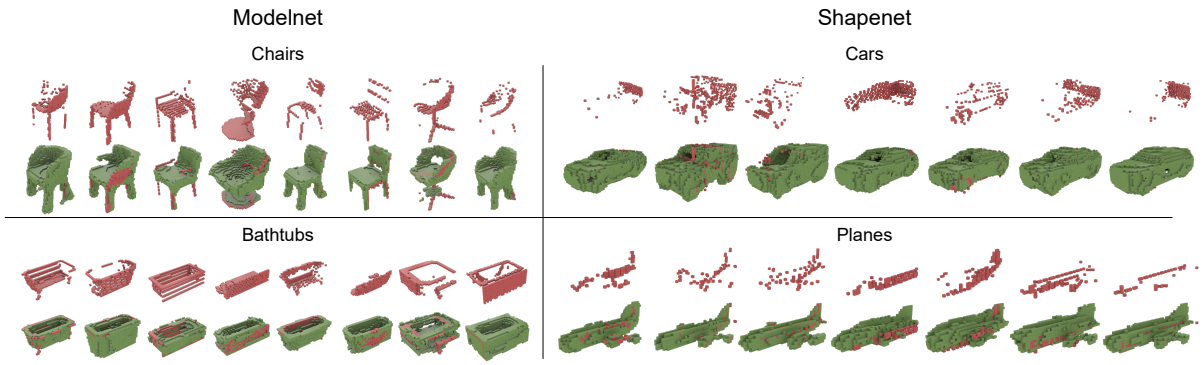
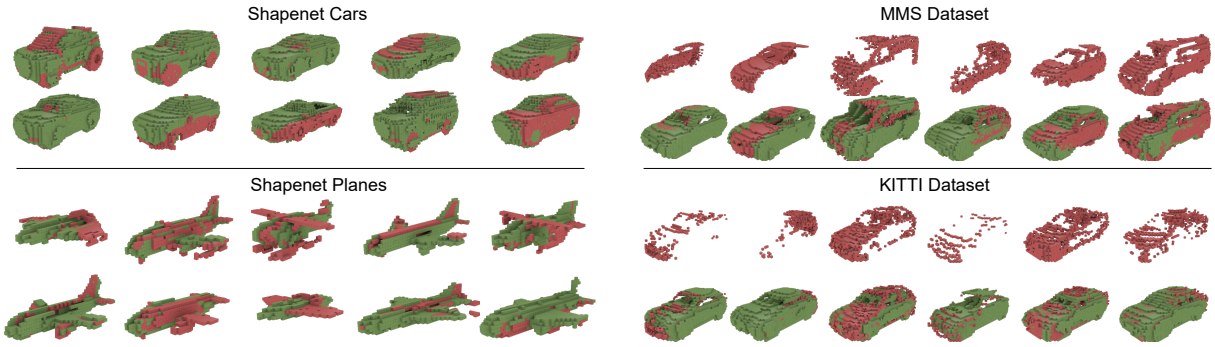



*Figure 8.15: Predicted shapes for ModelNet chairs (top left), bathtubs (bottom left), ShapeNet cars (top right), and ShapeNet airplanes (bottom right). All networks were trained on very sparse samples. Red voxels denote the input and green voxels the predictions.*

indicated by the occluded input example. Figure 8.16a shows additional results for high-resolution cars and planes from ShapeNet. These results are illustrate the variability of the dataset, and the ability of the GAN to complete various instances with very different shapes.




*(a)* Completion examples for *cars* and *planes*.

*(b)* GAN trained on MMS (top), applied to KITTI without retraining.

*Figure 8.16: Examples for synthetic data (a) and real data (b)*

To show that the model works also on real-world 3D data, cars from the MMS dataset were extracted. As described in 6.3, a set of 8941 fairly clean cars was obtained. The cars are heavily occluded, with at least one side entirely missing in most cases. In this case, the GAN has to cope with the fact that the shapes are only very roughly aligned and that the unsupervised extraction of the data also results in wrong samples. Figure 8.16b (upper row) shows the result on the MMS Dataset. It can be seen that the GAN is able to complete the shapes just as well as in the synthetic experiments.

Figure 8.16b (lower row) shows predictions for the KITTI dataset, generated with the GAN trained on the MMS dataset, which features a completely different type of laser scanner with a much higher point density. The model generalises very well, one can see that also for KITTI data, the predictions are mostly complete and match the input observations (i.e., the model does more than returning a mean car shape). Failures, where the generator predicts a severely incomplete shape or one that is in significant disagreement with the input, occur mostly on rare, big shapes, like trucks or buses.

The Results in Figure 8.17 and 8.18 show outcomes from a model that was trained on the MMS dataset. The images were created by applying different thresholds to the occupancy grid. If $G(x) < t$, a voxel is considered as being occupied, else it is considered being free, where $t$ is the threshold. The threshold was normalized to between 0 and 1 by the minimum and maximum value the generator is predicting. Figure 8.17 shows the outcomes for a typical car in the mobile mapping

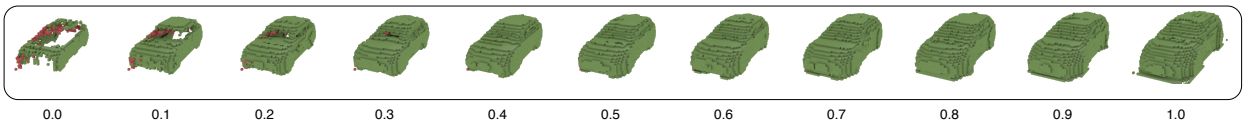



*Figure 8.17: Example that shows how the threshold value for deciding whether a voxel is occupied or not affects the result. The corresponding threshold t is shown below each car.*

dataset. Using a low threshold the car is very sparse and has holes in the window. When the threshold is increased, the car size grows and the holes in the windows become closed.

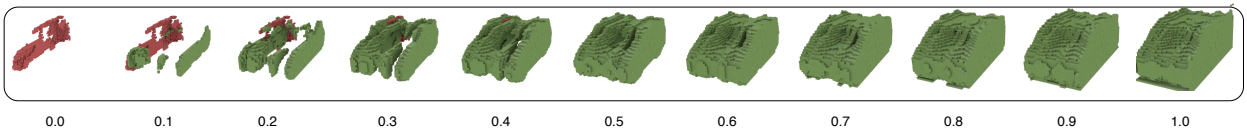



*Figure 8.18: Example that shows how the threshold value for deciding whether a voxel is occupied or not affects the result. The corresponding threshold t is shown below each car.*

As can be seen in Figure 8.18, the generator has sometimes problems to complete larger cars. The reason for this might be that there are not enough large cars in the dataset. If the threshold is below 0.5, the generator prediction seems to contain a mixture of two cars, one in the middle and one on the outside. If the threshold rises above 0.5, the cars are connected and form a larger car.

### 8.2.4 Conclusion and Discussion

In this section, the method for self-supervised adversarial completion of partially observed 3D shapes was tested. The prerequisite is that the object class is known and has moderate shape variability so that it can be roughly aligned within a bounding box.

It was shown qualitatively and quantitatively that the proposed GAN is able to complete objects from incomplete observations only. The quantitative results show that the GAN is positioned approximately between the baseline and the fully supervised method and is sometimes close to the semi-supervised method. An interesting direction for future work is to include attributes of the surface points beyond their location, such as color or normal vectors, perhaps even certain material properties. It might also be useful to move away from voxel representation and predict complete shapes in the form of point clouds or surface meshes.

Finally, it is important to test the extent to which shape completion affects label transfer. This is an important open question for further research. It is conceivable that estimating occupied voxels in unknown regions positively influences label transfer when using ray tracing. However, integrating this method into the overall system presented is not trivial and requires further research and development.

# 9 Conclusion and Discussion

The goal of this work is to minimize the time-consuming labeling of 3D point clouds by learning to transfer labels from images classified by DCNNs which are pretrained on publicly available datasets. To this end, this thesis presents a method to transfer labels from 2D multi-view images into 3D. To transfer labels from images to 3D point clouds a dataset consisting of more than 15 billion laser scan points and 250 thousand 2D camera images were acquired using a fully calibrated mobile mapping system. All images were semantically segmented using a publicly available pretrained DCNN. By projecting each 3D point into the associated camera images, the 2D pixel predictions could be mapped to the corresponding 3D points. The problem with naive mapping is that it ignores various errors, such as calibration errors, occlusions, prediction errors and incompatible label policies. The incorrect 3D-to-2D association leads to a wrong class label assignment in 3D (label noise). Within this dissertation various error sources were discovered, analyzed and tried to be compensated. In the following the proposed methods to tackle these problems are summarized and compared to each other.

## 9.1 Summary and Discussion

First, all MMS images were semantically segmented using Deeplabv3+ pretrained on Cityscapes. The naive baseline was investigated in Chapter 7.2. By comparing the results on the MMS images to a manually annotated reference set, it was shown that Deeplabv3+ only achieved a mIoU of 0.661 on the MMS image dataset (see Section 6.1.1) compared to the mIoU of 0.818 on the Cityscapes test set. This domain gap is assumed to be due to the different camera position, sensor model, and an unknown environment. Then, all classified images were mapped to the corresponding point clouds using the naive label transfer with majority voting. By comparing the transferred labels in 3D with another manually annotated reference set, it was shown that this resulted in further degradation of label quality, leading to the **baseline** result with a mIoU of **0.481**. By comparing the confusion matrix before and after label transfer, several types of noise were identified, namely calibration errors, dynamic occlusions, label policy errors, and regular and self-occlusions.

### 9.1.1 Scanstrip-Based Label Error Correction

To improve the mIoU several scanstrip-based methods were proposed. First a GBDT was trained that learns to map from single 3D point features to the reference set. By using all available point features, including the proposed campaign count $\xi$ and the class histogram $h$, the GBDT achieved an mIoU of 0.557. Subsequently, the Scanstrip Network (SNet) was introduced which significantly increased this result by achieving an mIoU on the same testset of 0.667. The network was compared to different state-of-the-art semantic segmentation networks such as HRNet (0.612-0.628 mIoU) or Deeplabv3+ (0.583-0.620 mIoU) and also to similar network architectures like FCN (0.549-0.606 mIoU) and U-Net (0.614-0.63 mIoU). Regardless of the training strategy or the chosen hyperparameters SNet achieved the highest mIoU among them.

To further improve the results, a semi-supervised scanstrip-based correction was introduced. The basic idea was to pretrain SNet firstly on large sets of scanstrips, using automatically generated noisy labels as reference. Secondly the network is fine-tuned on the actual reference dataset. As the

normal SNet already uses the noisy labels as input, the pretraining would lead to a trivial solution where SNet simply learns the identity function. Therefore, SNet has been heavily modified so that it learns the representation of all remaining features except the histograms, and also extracts a high-level feature representation that can be used in the second training phase. This approach reached so far the best results with an overall mIoU of **0.709**.

### 9.1.2 End-To-End Multi-View Label Transfer

To solve the complete label transfer from 2D to 3D, the Multi-View Network (MVNet) and Label Transfer Network (LTNet) were introduced. The first network works exclusively in 2D, before the transfer. It will be summarized in the first part of this section. The second part summarizes LTNet, which is able to solve the label transfer end-to-end.

**Multi-View Network (**MVNet)

In order to be able to correct the labels at an early stage, a network was developed that can be trained on multi-view images. The problem is that, especially in computer vision, grid-like data structures are preferred, which makes it difficult to learn from multi-view image data that can occur in any quantity and order. For this purpose, Multi-View Network (MVNet) was introduced, which is capable of linking an arbitrary length of multi-view image predictions. The network was trained to map a list of noisy 2D image labels to the 2D reference set. Here, the noisy predictions form the baseline with an mIoU of 0.709. The multi-view network was compared to Single-View Networks (SVNets). This network is the same as MVNet, but without the ability to relate predictions. Regardless of the hyperparameters or trials chosen, MVNet outperformed SVNet in almost every case, confirming that this could only be achieved by the ability to relate predictions for multiple images. Although the networks only had access to an equivalent of 3-4 labelled images, MVNet achieved a mIoU of 0.784 compared to SVNet with a maximum mIoU of 0.735 without label propagation and a mIoU of 0.734 with label propagation. Note that these results are for 2D images, so the mIoU is calculated *before* the label transfer into 3D. The results are not directly comparable to those in Subsection 9.1.1, because the two reference sets belong to different domains.

Since early predictions errors by Deeplabv3+ contribute to the label noise in the 2D-to-3D transfer, it was tested if the predictions of MVNet can be used as pseudo-labels to fine-tune Deeplabv3+ on the MMS-images. The approach was tested against different other approaches, all of which have access to the same MMS-image subset and the same ground truth labels as MVNet. Also the validation and testing sets were chosen so that all results are comparable to each other and that they do not intersect with the training of MVNet. Finally Deeplabv3+ fine-tuned on the MVNet pseudo-labels increased the mIoU from 0.632 to 0.696. Using naive fine-tuning or the approach described by Zhu et al. (2020) yielded only an mIoU of 0.628 and 0.663. Please note that these results are not directly comparable to the ones achieved by MVNet in the in the preceding paragraph since both test sets are different.

**2D-to-3D Label Transfer**

Finally, it was shown that by combining MVNet with the pretrained SNet to form Label Transfer Network (LTNet), the full label transfer from 2D to 3D can be learned. This network achieved a mIoU of **0.749** compared to the previous best performing model SNet with a mIoU of **0.709**. The network improved the IoU in almost all classes. Additional ablation studies showed that each subnetwork alone is unable to achieve similar performance, and that they could only achieve their full potential when the two networks were combined, confirming the observations made in the previous sections for MVNet and SNet. Note that care was taken to ensure that both LTNet

and Scanstrip Network (SNet) are directly comparable, as both networks have exactly the same training, validation, and testing sets.

### 9.1.3 Self-Supervised Completion

In Sections 8.1 and 8.2, two GANs capable of predicting photorealistic images and complete shapes were tested. In the scope of this thesis they can be used to handle dynamic occlusions and self-occlusions during the label transfer. First, the CGAN from the Section 5.1 was tested in several ablation studies. The network is able to predict photorealistic images from point clouds only. By using the date of acquisition as input, the seasonal characteristics of the output became controllable. This means that for each point cloud a photorealistic image can be predicted for each season. This was also confirmed quantitatively by calculating FID and MS-SSIM scores for each campaign. The ability to generalise could be demonstrated on point clouds from another city (Karlsruhe) that were never part of the training set (Hannover). Most importantly, it was shown that the synthesised images can be successfully semantically segmented by a pretrained Deeplabv3+. However, some classes, especially those with small physical size or rare classes, were not recognised as well. More interestingly, the synthesised cars were qualitatively all successfully recognised in their correct locations, which was also quantitatively confirmed in later studies. Finally, the synthesised images were tested for 2D-to-3D label transfer. To do this, they were stitched together to have the same resolution as the original MMS images. Then the naive label transfer was compared once with a set of real images and once with a corresponding set of fake images, all generated for the same camera positions as the real images. Figure 8.11 showed the results after the label transfer by comparing the mapped labels with the 3D reference set. For the most prominent classes, such as *road*, *building*, *sidewalk* and even *pole*, both approaches were quite similar. But more importantly, using the CGAN images gave a significant increase in the *car* class, which otherwise suffers from typical dynamic occlusion errors. The images in Fig. 8.12 quantitatively confirmed that when the synthetic images were used, typical problems such as roads being assigned to the *car* class did not occur. However, parked cars were still well detected, showing that the CGAN was able to synthesise cars matching the point cloud.

The final experiments in Section 8.2 showed that it is also possible to learn to complete shapes in a self-supervised way. For this, several datasets with incomplete 3D shapes were created. First, naive label transfer was used together with region-growing to extract $>$ 9,000 3D car scans that were roughly aligned to a car template using ICP. In addition, incomplete cars were extracted from KITTI using the annotations of the dataset. As the real (incomplete) 3D scans do not contain a corresponding ground truth or a complete scan, several other datasets were created for which ground truth is available. For this purpose, car, chair, plane and even bathtub meshes were extracted from the synthetic datasets Shapenet and Modelnet. By simulating self-occlusions with the same procedure as presented by Stutz and Geiger (2018), realistic incomplete 3D scans could be created. In these datasets, the ground truth is given by the mesh itself, which is not available to the GAN during training and testing. The qualitative evaluation showed that the GAN is indeed able to complete the incomplete scans in a meaningful way. This means that the GAN reacts correctly to the (incomplete) input by predicting a suitable shape and that the gaps are filled in a meaningful way. By applying the pretrained GAN to the KITTI scans, it was also shown that it generalises to a different dataset of scans acquired by a different laser scanner model. The quantitative evaluation in Table 8.4 confirmed this observation. Here, the network was compared to the approaches of Dai et al. (2017) and Stutz and Geiger (2018), which used (semi-) supervised approaches. A prior shape was qualitatively selected as the baseline for each dataset as a typical complete example. This was compared to all other examples in the dataset, simulating a GAN that does not respond well to the input and simply outputs the same shape. As all shapes were voxelized

the mIoU between ground truth and generated shape could be compared. The results showed that the GAN performed significantly better than the baseline and almost as good as the approaches using supervision. For example, using the synthetic car scans the baseline achieved an mIoU of 0.64 and the GAN an mIoU of 0.7, see Table 8.4. The (semi-) supervised methods achieved an mIoU of 0.78 (Stutz and Geiger, 2018) and 0.87 (Dai et al., 2017). Another example with ModelNet chairs illustrates the performance of the GAN, because these shapes are much thinner and a deviation is therefore punished more severely. Here the prior shape achieved an mIoU of 0.15 which is even worse than using just the incomplete sample with an mIoU of 0.21. The GAN however was able to get an mIoU of 0.33 against the mIoU of 0.41 (Stutz and Geiger, 2018) with $\leq 10\%$ supervision and 0.61 (Dai et al., 2017) with 100% supervision.

### 9.1.4 Conclusion

Seven research hypothesis were presented in the introduction, all of which describe statements that outline the entire thesis regarding 2D-to-3D label transfer. In the baseline, the causes of label noise in 2D-to-3D label transfer were identified and analysed in detail (hypothesis 1). With GBDT and SNet the label noise can be corrected after aggregation. It was shown that SNet outperforms several other state-of-the-art networks even with only a few reference labels available (hypothesis 2). Subsequently, SNet was adapted for semi-supervised training by using large amounts of noisy labels for pretraining and only a small amount for fine-tuning (hypothesis 3). By linking multi-view images, the prediction performance can be increased in 2D compared to single-view images. In addition, by using the pseudo labels generated by MVNet, the domain gap on the MMS images for Deeplabv3+ can be reduced (hypothesis 4). The highest mIoU in 3D was achieved by combining all previously presented methods in LTNet. The network was able to learn a full multi-view 2D-to-3D label transfer as described in the hypothesis 5. In the final experiments, it was shown that two causes of label noise could be handled without any ground truth. First, the CGAN was able to handle dynamic occlusion by synthesizing photorealistic images from point clouds that served as an interface for the pretrained DCNN (hypothesis 6). The other GAN was able to predict complete shapes from incomplete 3D scans without ever seeing a complete object (hypothesis 7). Since it is difficult to treat areas behind self-occluded objects with ray tracing, the GANs predictions could further reduce label noise in label transmission by blocking rays that would likely hit an occupied voxel.

## 9.2 Outlook

Within the thesis many different methods were introduced in order to semantically segment 3D point clouds with only very few ground truth labels available. All methods potentially opening branches for further research. As the late correction was done on scanstrips using 2D convolutions it would be interesting to analyze how 3D-based ANNs would perform. For example the mapping from noisy label to the reference set could be learned in 3D using KPConv (Thomas et al., 2019) instead of using a 2.5D representation. It would be interesting to see if in this way the whole process including LTNet could be adapted.

In terms of MVNet, it would be very interesting to analyse how the network behaves and performs with many more ground truth labels. The network could be compared to modern semantic segmentation networks such as HRNet or Deeplabv3+ if more training data were available. The network architecture could also be improved by modifying it to semantically segment entire patches instead of only classifying the central pixel of each patch, which would probably lead to more homogeneous results.

LTNet uses two sub-branches, one for the 2D domain and one for the 3D domain, both of which are interchangeable with any other architecture. For example the 3D sub-branch of LTNet could be replaced by a state-of-the-art 3D semantic segmentation or classification architecture. The results revealed that LTNet suffered from inhomogeneous predictions, because it predicted classes point-by-point. Instead the network could make a prediction for a whole 2.5D patch or 3D region at once. When creating the architecture of LTNet, it was noticed that the symmetric function for aggregating the multi-view observations in the 2D subbranch is very important. In the current implementation, average pooling is used for this purpose. It would be interesting to use methods for example from the field of natural language processing, which often work with sequence-to-sequence models (Vaswani et al., 2017). Finally, the 3D subbranch of LTNet was pretrained and not tuned during training. Since the multi-view subbranch was initialised randomly, it would be interesting to use a pretrained network using the 2D reference set here as well, which could further improve the label transfer.

As already discussed, the CGAN could also be improved to generate better images. First the architecture could be changed similar to ACGAN by (Odena et al., 2017). Here, the discriminator would also have access to the information of the date of capture, which would force the generator to react appropriately to seasonal characteristic, which could further improve the prediction quality. It would also be very interesting to further combine the training of MVNet and the CGAN. As the GAN can generate images for any viewpoint in any season, the MVNet would have access to any multi-view viewpoint for training. This could be used to further improve the label transfer. Finally, the real goal for the CGAN is to mitigate dynamic occlusions in label transfer. However, it was never explicitly defined that the network should ignore dynamic objects. The campaign count $\xi$ provides information about how dynamic each point is. By using this metric, one could force the CGAN to ignore dynamic 3D points, which could result in images that better match the point cloud, thus improving 3D label transfer.

Finally, the treatment of self-occlusions could be further investigated. It would be interesting to see to what extent the predicted complete shapes would help in the label transfer step. To do this, the predictions need to be mapped into the global point cloud and the ray tracing needs to be adjusted accordingly to account for the predictions by the GAN. This process will likely need to be carefully tuned to account for the soft predictions for each voxel cell. Since the car scan extraction process allows any other object to be extracted, it would be interesting to see how the GAN is able to complete different types of real objects, such as pedestrians, bicycles, vegetation or even buildings. However, for larger objects, the network architecture needs to be changed significantly, as the current implementation only allows voxel grid representations, which are computationally not very efficient. It would be interesting to see if it is possible to predict raw point clouds similar to PU-GAN by Li et al. (2019).

## List of Acronyms

ACGAN Auxiliary Classifier GAN . . . 45
ANN Artificial Neural Network . . . 1
CCD Charge Coupled Device . . . 7
CGAN Conditional Generative Adversarial Network . . . i
CMOS Complementary MetalOxide Semiconductor . . . 7
CRF Conditional Random Field . . . 46
DCNN Deep Convolutional Neural Network . . . 1
DEM digital elevation models . . . 16
DMI Distance Measuring Instruments . . . 13
DNN Deep Neural Network . . . i
FCN Fully Convolutional Network . . . 36
FID Fréchet Inception Distance . . . 127
GAN Generative Adversarial Network . . . 32
GBDT Gradient-Boosted Decision Trees . . . 55
GNSS Global Navigation Satellite System . . . 1
GPU Graphics Processing Unit . . . 29
ICP Iterative Closest Point . . . 13
IMU Inertial Measurement Unit . . . 1
IoU Intersection over Union . . . 59
kd-tree k-dimensional tree . . . 15
LiDAR Light Detection And Ranging . . . 10
LTNet Label Transfer Network . . . i
mIoU mean Intersection over Union . . . i
MMS Mobile Mapping System . . . 13
MS-SSIM Multi Scale Structural Similarity . . . 127
MVNet Multi-View Network . . . i
RNN Recurrent Neural Network . . . 31
SAPOS Satellite Positioning Service . . . 77
SLAM Simultaneous Localization and Mapping . . . 13
SNet Scanstrip Network . . . i
SVNet Single-View Network . . . 61
ToF time-of-flight . . . 11
UAV Unmanned aerial vehicle . . . 13
UTM Universal Transversal Mercator . . . 81
WGS 84 World Geodetic System 1984 . . . 13

# List of Figures


1.1 An example showing three different images of the same scene. The red star marks the same spot in these images for which a prediction was made by 2D DCNN. Although all three pixels are in different positions in different images, they belong to the same object (wall). By linking the three predictions, the wrong *car* prediction can be detected and corrected to *wall*. . . . 2

2.1 A camera sensor measures the RGB values using a Bayer pattern (left). The values are stored in the pixel coordinate system (middle). The projected image on the sensor is given in the image coordinate system (right) . . . . . 7
2.2 Schematic relationship between a point in camera coordinates and the image plane . . . . 8
2.3 Two images showing a pair of real laser scanners (left) and a schematic drawing of a time-of-flight (ToF) laser scanner (right) . . . . . 10
2.4 The LiDAR measures the range, horizontal and angular rotation given directly in a spherical coordinate system . . . . . 12
2.5 Helical line scanner pattern (points) with scanner continuously rotating 360° along $\phi$ (Eq. 2.8) while moving in $z$ direction. The image corresponds to the coordinate system in Figure 2.4b with fixed $\theta = 90°$. Each red line shows a laser beam after the scanner has been rotated 396° and moved along the z axis. In total, the scanner head is rotated 5.5 times in this plot. 13
2.6 3D point cloud visualization of a car scanned with the RIEGL VQ-250 using the VMX-250 mobile mapping system. . . . . 14
2.7 Different voxelizations of the point cloud of the car shown in Figure 2.6. The volume size is given by $r$, the density by $\rho$, and the edge length by $e$. . . . . 15
2.8 Example scanstrip captured with a line scanner mounted on a Mobile Mapping System (right) and the corresponding coordinate system (left) . . . . . 17
2.9 Example of a very simple decision tree with three levels and three attributes (Color, Shape and Size). The tree is trained on a binary classification dataset and outputs either Yes (1) or No (0). . . . . 20
2.10 Example of a simple multilayer perceptron (MLP) with two fully connected layers and two weight matrices $W^1$ and $W^2$. . . . . 26
2.11 An example for a 2D convolution without padding. The input is $3 \times 6 \times 1$, the kernel size $2 \times 2 \times 1 \times 1$ and the stride is $s_1 = 1$,$s_2 = 1$. The output $g$ has the size $2 \times 3 \times 1$ . . . . . 29
2.12 Mapping an input sequence $X$ of length $n$ to the output sequences $R$ and $S$ of size $n \times s$. On the left an RNN is shown that maps from an input to a hidden state with matrix $U$, from hidden to hidden states with $W$, and from the hidden states to the output with matrix $T$. On the right, each element of the sequence $X$ is mapped independently to the matrices $Q$, $K$, and $V$ using three MLPs. $Q$ and $K$ are used to calculate an attention map $A$ that relates all entries in $V$, resulting in the output $S$. Both methods are similar in the way that they relate input sequences to each other. However, they do not produce the same results and are not equivalent to each other . . . . . 30
2.13 The general structure of generative adversarial networks. The generator learns the mapping $x = G(z)$ and the discriminator learns to classify whether $G(z)$ and $x$ are real or generated. 32

3.1 Number of occurrences for the search term “semantic segmentation deep learning” returned by `scholar.google.com` for each year. The search excluded patents and citations. . . . . 35
3.2 Schematic overview of two semantic segmentation networks . . . . . 36
3.3 Schematic overview of the Deeplabv3+ architecture. Image source: Chen et al. (2018) . . . 37
3.4 Schematic overview of the connections in HRNet. Image source Sun et al. (2019). . . . . 38
3.5 Schematic overview of point net. Image Source (Qi et al., 2017a) . . . . . 40
3.6 Examples for successfully corrected predictions after retraining of HRNet as presented by Peters et al. (2020). The figure shows, from left to right, the input image, the uncorrected prediction, the corrected prediction, and the manually labelled ground truth. . . . . 47

4.1 The diagram shows the general procedure of label transfer and correction process, described in this chapter. (1) First the data is acquired using a fully calibrated MMS. (2) Secondly all matches between 3D points and 2D pixels are calculated by projecting the 3D points into the corresponding images. (3) These images are pixelwise classified by a pretrained DCNN, where different colors indicate different predicted classes and different shapes represent different object classes. Note the (magenta) cube in boxes 3 and 4 indicates a wrong assignment between pixel and 3D points. (4.I) By learning how to transfer the labels from 2D to 3D, the point cloud is correctly labelled. (4.II) By observing the predictions from several views in 2D, it is learned how to correct the initial predictions. . . . . 49
4.2 A point cloud with examples for regular and self-occlusions. The tree and car are once colored using the nearest RGB value and once colored using ray tracing. . . . . 51
4.3 A point cloud colored according to the point reflection (left), the RGB color (middle) and campaign count $\xi$ (right). Although the the color was assigned using ray tracing, the cyclist in the center of the images is incorrectly colored. . . . . 51
4.4 Aligned point clouds in which each color indicates a different mapping campaign. Image source Schön et al. (2018) . . . . . 52
4.5 Schematic example of the point cloud annotation process. The labels on the point cloud are assigned with majority vote. . . . . 53
4.6 An example of label noise due to Cityscapes labeling policy. The left image shows a semantically segmented MMS image. The right image shows a point cloud with classes assigned to the majority vote, where the facade was incorrectly assigned to *vegetation*. Image source (Peters and Brenner, 2019). . . . . 53
4.7 Histograms clustered with k-means clustering. Classes are randomly colored. The red points on the building correspond to histograms containing *vegetation* and *building* labels. Image source (Peters and Brenner, 2019). . . . . 54
4.8 Scanstrip Network (SNet). The network follows a U-Net structure (Ronneberger et al., 2015) with residual blocks. The kernel size is $3 \times 3$ in every convolutional layer. The dotted lines indicate the skip connections between the layers. The size of the feature channel is $B$ and the number of predicted classes is $C$. . . . . 55
4.9 An image showing the Scanstrip network $\text{SNet}_{SSL}$ in the first training phase of the semi-supervised learning approach. The input of the network are all features except the label histogram $h$. Here the label histogram serves as ground truth. The classification head is later used in the second training phase as seen in Fig. 4.10 . . . . . 56

4.10 In the second phase of the semi-supervised approach, all layers except the last three (dark green and light red) are frozen. The input of the last layers are the scanstrip containing only the histograms and the extracted features of the last four layers of the network from step one. All inputs are concatenated as shown in the figure and then passed to three successive convolutional layers, each of which has kernels of size $1 \times 1$ . . . . . 57
4.11 Procedure for creating an input list for one 3D point. The input data are multi-view images (Box 1). Different colors in Box 2 indicating different classes per pixel. In Box 3 one 3D point $p_j$ is mapped to all images creating $n_j$ correspondences (corr). The resulting list $z_j$ for the 3D point $p_j$ with length $n_j$ is the input of MVNet. The length of the list corresponds to the number of multi-view images. Each entry contains the pixel wise prediction $\hat{d}_{n,j}$ for each of the $n$ images, as well as 2D image features and RGB image patches cropped around each corresponding image pixel. . . . . 60
4.12 The proposed network architecture for learning to predict a corrected list of labels with $C$ classes for a list of multi-view observations $z_j$. The weights of the dense layers (yellow and green blocks) are shared. However, they are applied to each list entry separately. The same is true for the ResNet networks (Blue triangles), they are applied to each image patch individually, but all their weights are shared. The joint output of all ResNet networks is an $n_j \times 1 \times 1 \times 128$ tensor with $1 \times 1 \times 128$ per image, which is then reshaped to $n_j \times 128$. All dense layers in the network use batch normalization and ReLU as activation. . . . . 61
4.13 The ResNet block from Figure 4.12. This subnet repeatedly uses residual blocks with a striding of two until the input image patch is encoded in a feature vector of size $1 \times 1 \times 128$. The residual blocks are described in the right part of the image, where BN stands for batch normalization. . . . . 62
4.14 A schematic overview of the Label Transfer Network (LTNet). The upper part of the network corresponds to the MVNet shown in Figure 4.12 up to the penultimate layer. The difference here is that MVNet receives also a list of 3D point features $v_j$. The extracted features of the MVNet are reduced to a fixed-size vector using average pooling. The lower part of the network is the same as in the semi-supervised scanstrip network ($\text{SNet}_{SSL}$), Figure 4.10. $\text{SNet}_{SSL}$ is pretrained and the weights are frozen during the training of LTNet. The feature of the penultimate layer of the $\text{SNet}_{SSL}$ corresponding to the 3D point $p_j$ is appended to the reduced multi-view feature vector. Both features are then used in conjunction to predict the final class for $p_j$ . . . . . 63

5.1 This diagram shows an extension to Figure 4.1. In this Figure the MMS is mainly replaced by a GAN which estimates multi-view and multi-modal images based on the measured point clouds by the LiDAR. Additionally a step for resolving self-occlusions is added in the end. . 65
5.2 The generator network receives a projected point cloud image with two channels and creates a photorealistic RGB-image. A one-hot encoded season vector of size $S$ is passed to a fully connected layer (red), whose output is reshaped to $1 \times 1 \times 1024$ and concatenated to the bottleneck (blue). . . . . 67
5.3 The multi-scale discriminator networks. . . . . 68
5.4 Methodology: Red/blue and green color indicate real and generated data, respectively. Orange blocks are merged voxel-wise between real and synthesized, indicated by the "max" operation. The Encoder $\Theta(\cdot)$ will be described in the network architecture section. It encodes each subregion into a vector which are then average pooled to a fixed size representation and passed to the discriminator. . . . . 71

5.5 First, a voxelized shape is taken as input (left image). Then, the input is divided into $n = n_x \times n_y \times n_z$ blocks (middle image in BEV). Each block is labeled as "real" (red) or "incomplete" (blue) based on the point density in each block. . . . . 72

5.6 This figure shows an example of network $\Theta(\cdot)$. It encodes a block of size $n_x = 8 \times n_y = 16 \times n_z = 8$ into a feature vector of size $1 \times 1 \times 1 \times 256$. . . . . 72

5.7 The generator network receives an incomplete shape (red) as input and completes it. The network receives a voxel grid of size $32 \times 64 \times 32$ which is passed to a sampling layer (violet) that converts the grids to a point cloud. This point cloud is then passed to three PointNet++ set abstraction layers (green) that follow exactly the original implementation by Qi et al. (2017c) with the same hyperparameters. The encoded shape is then reshaped and passed to a series of 3D transposed convolutions (yellow) until the original size of $32 \times 64 \times 32$ is reached. The kernel size is $3 \times 3$ for the convolutional layer. . . . . 74

6.1 Route of the MMS vehicle for the mapping campaign. The city center of Hannover is in the lower right corner and the Leibniz University is in the middle of the image. . . . . 77

6.2 Examples of semantically segmented MMS images of the same scene with two different viewing angles. The images on the left show the original MMS image. The images on the right show the corresponding predictions from Deeplabv3+. The MMS captures images pointing to the rear of the vehicle. The rows show the two available viewing angles. . . . . 78

6.3 23 human annotated images. The original images are a subset of the MMS dataset. All images are labelled according to the official Cityscapes guidelines. . . . . 79

6.4 Examples of different visualizations of the same scanstrip (images are cropped). Left: Results of the segmentation using the method described by Brenner (2016) (each segment randomly colored). Middle: Distance measured by the laserscanner. Right: Intensity measured by the laserscanner. . . . . 80

6.5 The images show examples of two labeled scenes. Each scene is shown once as annotated in the scanstrip (a and c) and as a 3D point cloud (b and d). . . . . 80

6.6 The Scheme shows the MapReduce approach for rendering large point clouds. Image source (Peters and Brenner, 2020). . . . . 81

6.7 Each row shows the same scene. The images on the left show the real camera images. The images in the middle show the projected point cloud colored by reflectance and the images on the right are colored by the distance. . . . . 82

6.8 Examples for dynamic occlusions. The red car and the bicycle rider appears in the camera image (left) but not in the point cloud (right). . . . . 83

6.9 The same scanstrip scene colored by reflectance (left), class labels (center) and segment ID (right). . . . . 84

6.10 3D street scene which shows only points classified as car, colored randomly by instance (before and after filtering of outliers) and a prototype car used for orienting incomplete scans (right). 84

6.11 Example of an extracted car before (left) and after voxelization (right). The voxelized point cloud shows only the center of each voxel . . . . . 85

6.12 Examples of self-occluded and voxelized objects from the Shapenet dataset. Left: A complete car and the result of the simulated self-occlusion. Right: A plane and its self-occluded counterpart. All voxels are colored by height, where green is low and red is high. . . . . 86

7.1 An example which shows the prediction quality of Deeplabv3+ on the MMS-images. The Figure shows the input (left), the semantically segmented image (middle) and the corresponding ground truth image (right). . . . . 88
7.2 Deeplabv3+ baseline: The figure shows the IoU per class on Cityscapes (Deeplabv3+$_{Orig}$), on the MMS-images (Deeplabv3+$_{2D}$) and after the naive label transfer into 3D (Deeplabv3+$_{3D}$). 89
7.3 Confusion matrices for Deeplabv3+ before (left) and after mapping into 3D (right). Both matrices are normalized per row. . . . . 90
7.4 Examples for different types of labelling errors: The first image shows dynamic occlusions (Ellipse 1). The Second image shows calibration errors (Ellipse 2) and Label-Policy errors (Ellipse 3) . . . . . 90
7.5 An Example for self-occluded cars (blue), as seen from above. . . . . 91
7.6 Detailed results for the different variants of the gradient boosted decision trees in comparison to the baseline *Deeplabv*3+$_{3D}$ . . . . . 94
7.7 Loss on training set (left) and mIoU on the validation set (right). The arrows in the right plot indicate the global maximum. All plots were smoothed with moving average for better visibility. . . . . 96
7.8 The Loss of SNet$^{256}_{SSL}$ in the second training phase (left) and mIoU on the validation set (right). 99
7.9 A comparison of the IoU on the test set for every class between the naive label transfer (blue), supervised correction (orange) and semi-supervised correction (green) . . . . . 100
7.10 Qualitative comparison of the presented methods for three different scenes. Each column represents a scene, which is colored differently per row according to the results of the individual methods. . . . . 102
7.11 Comparison of the naive result to SNet$^{(256)}_{SSL}$ in a scene shown in Fig. 7.4 containing errors due to dynamic occlusions. . . . . 103
7.12 Comparison of the naive result to SNet$^{(256)}_{SSL}$ in a scene shown in Fig. 7.4 containing label policy and calibration errors. . . . . 103
7.13 Comparison of the naive result to SNet$^{(256)}_{SSL}$ in a scene containing regular- and self-occlusions. 103
7.14 Cars and bicycles extracted in the scene shown in Figure 7.13. . . . . 104
7.15 Three point clouds show the scenes in which the MMS images were acquired for training and testing. The images from scene 1 are used for training. The images from scene 3 are used for validation and testing. All point clouds are colored according to reflectance values. . . . . 106
7.16 Schematic overview visualizing the creation of the train, validation and test set for the Multi-View Network (Associated). The remaining annotations are later used in the fine-tuning step of the DCNN (leftover). . . . . 107
7.17 Two grids showing the mIoU for the models using self-attention vs. no self-attention. The networks were trained using the **Tversky loss** with $\alpha = 0.5$ and $\beta = 0.5$ with different batch (rows) and window sizes (columns). . . . . 110
7.18 Two grids showing the mIoU for the models using self-attention vs. no self-attention. The networks were trained using the **Dice loss** with different batch (rows) and window sizes (columns). . . . . 110
7.19 Two grids showing the mIoU for SNet trained on the propagated labels (trial three). These models did not use self-attention. All ablations were done once for Dice and once for Tversky loss. . . . . 110

7.20 Comparison of $\text{MVNet}_{\text{DL}}$ and $\text{SVNet}_{\text{DL}}$ with the baseline shown in Figure 7.2. The height of each bar for each class is given by the mean across all experiments shown in Figure 7.18 and 7.17. The black error bars show the standard deviation for each class within all ablation experiments. Please note the results for Deeplabv3+$_{2D}$ are obtained using the total reference set and are therefore not directly comparable to the ones of $\text{MVNet}_{\text{DL}}$ and $\text{SVNet}_{\text{DL}}$. They are intended to show in which cases label transfer from 2D to 3D lead to an increase or decrease of IoU. The actual improvement to the Deeplabv3+ baseline is shown in the next subsection. . . . . 111

7.21 Comparison of the baseline result for Deeplabv3+ before and after correction by SVNet or MVNet, both using the same hyperparameters. All results were obtained using the same test set. . . . . 112

7.22 Qualitative comparison of the original predictions by Deeplabv3+ (middle) and the predictions by $\text{MVNet}_{DL}$. The left column shows the original images. The column in the middle shows the predictions from Deeplabv3+ if a 3D point is available. The right column shows the corrected predictions by $\text{MVNet}_{DL}$ . . . . . 113

7.23 An example of an error case for MVNet. The first image shows a traffic-calmed road (red ellipse), correctly classified by Deeplabv3+ (middle image) as *road* (purple) and by $\text{MVNet}_{DL}$ (right image) predominantly as *sidewalk* (pink). . . . . 114

7.24 Randomly sampled pairs of images and labelled images created by MVNet using the merge strategy from Equation 7.10. The RGB images show the input data and the images to the right shows the predictions by MVNet which are used as pseudo labels for fine-tuning of Deeplabv3+. . . . . 115

7.25 Results on the "leftover test set" (see Figure 7.16). The first column shows the input and the last column shows the ground truth data. The columns Deeplabv3+ show the pretrained model, Pseudo shows the model after training with pseudo-labelling and fine-tuned shows the pretrained model after training directly on the train-set. $\text{DL}_{\text{MV}}$ shows the fine-tuned model on the corrected dataset by MVNet. . . . . 117

7.26 Comparison of the IoU per class for all ablation studies. Note that class *person* is not included in the test set. . . . . 118

7.27 Randomly selected images from the MMS dataset except for the training, validation and test set. The first column shows the input for the networks. The columns Deeplabv3+, Self-Trained, Naive, and $\text{DL}_{\text{MV}}$ show the respective experiments. . . . . 119

7.28 mIoU on the test set for LTNet using no attention (left) or self-attention (right) in the multi-view branch. The networks were trained using **cross-entropy** with different batch (rows) and window sizes (columns). . . . . 121

7.29 Two grids showing the test-mIoU for the models using only multi-view (2D) features vs using only the scanstrip feature vector (3D). The networks were trained using cross-entropy with different batch (rows) and window sizes (columns). . . . . 122

7.30 Two confusion matrices for $\text{LTNet}_{w=0,bs=32}$. The left shows the confusion when only multi-view features (without 3D scanstrip information) are used. The right side shows the results when all features are used. The matrices are normalized per row. . . . . 123

7.31 Comparison of IoU per class between naive baseline (blue), semi-supervised scanstrip network (orange) and LTNet (green). . . . . 123

7.32 Qualitative comparison of the semi-supervised scanstrip network $SN_{SSL}$ presented in Chapter 4.2.2 with LTnet for 5 different scenes (rows). The left column shows the reflectance values, the middle column the predictions by $SN_{SSL}$ and the right column the predictions from LTNet. In total, four different scanstrips from the test set are shown, with rows two and three as well as five and six belonging to the same scanstrip. . . . . 125

8.1 Input image (reflectance, left), synthesized image (middle) and real image (right) . . . . . 130
8.2 Summer (middle) and winter (right) representation of the same input point cloud (left) . . 131
8.3 Randomly sampled pairs of predictions (left) and ground truth images (right) for every campaign. The pairs are sorted by campaign number (campaign 0 is top left and 13 bottom right). . . . . 131
8.4 Different representations for the same input point clouds. Each row shows an example from one campaign. The columns show the input (left), the different predicted Campaign (1-13) and the corresponding real image (right). . . . . 132
8.5 Image pairs of Karlsruhe in winter and the corresponding synthesized summer images. . . . 133
8.6 Cars successfully classified in synthetic images (bright grey) using Deepblabv3+ pretrained on PASCAL VOC. . . . . 133
8.7 The example shows two ways of stitching different synthetic images into one large image. The left image shows a single synthetic image with $512 \times 512$ pixels. The middle and right images show the same scene, but with a resolution of $2056 \times 2452$, created by stitching many synthetic images together. Image b was created by calculating the average RGB value of the overlapping windows, and for image c the overlapping windows were weighted with a Gaussian mask to achieve a smoother result. . . . . 134
8.8 Resulting mIoU for ablation studies conducted with and without self-attention on the semantically segmented synthesized CGAN images. The average mIoU using no attention is 0.407 and with self-attention the average mIoU is 0.457. . . . . 134
8.9 Comparison of the IoU values for predictions with Deeplabv3+ on the fake images and the same predictions corrected by MVNet trained with parameters $w = 32$,$b = 32$. The mIoU for Deeplabv3+ is 0.41 and the mIoU of MVNet is 0.481 . . . . . 135
8.10 Examples showing randomly sampled Deeplabv3+ predictions on synthesized stitched RGB images (Synthetic) before correction and after correction with MVNet. . . . . 136
8.11 Comparison of the results obtained when using naive label transfer from images to 3D for different image sources. Deeplabv3+$_{3D}$ is a reference that shows the results when real MMS-images were used as source, see Section 7.2. For synthetic images Deeplabv3+ with Xception-65 or 71 backbones is denoted by X65 or X71. When predictions are based on the synthesized images, this is denoted by (GAN). When Deeplabv3+ was fine-tuned with MVNet this is denoted by $fine$. . . . . 138
8.12 Three example point clouds (left) showing the effect of naive label transfer using MMS-Images (middle) or synthesized GAN images (right) . . . . . 139
8.13 The GAN learned to complete shapes (green voxels) from sparse observations only (red voxels) 141
8.14 Prior shapes that were used for the quantitative evaluation process. The pictures show from left to right: car, plane, chair and bathtub gathered from Shapenet and Modelnet. . . . . 142
8.15 Predicted shapes for ModelNet chairs (top left), bathtubs (bottom left), ShapeNet cars (top right), and ShapeNet airplanes (bottom right). All networks were trained on very sparse samples. Red voxels denote the input and green voxels the predictions. . . . . 144
8.16 Examples for synthetic data (a) and real data (b) . . . . . 144

8.17 Example that shows how the threshold value for deciding whether a voxel is occupied or not affects the result. The corresponding threshold $t$ is shown below each car. . . . . 145
8.18 Example that shows how the threshold value for deciding whether a voxel is occupied or not affects the result. The corresponding threshold $t$ is shown below each car. . . . . 145

# List of Tables


6.1 Table with the dates of the individual measurement campaigns. The time of day shows that some started in the morning, others at noon or at dawn. . . . . 78

6.2 The table shows the distribution of annotated classes in the images (Support 2D) and point clouds (Support 3D) of the MMS-dataset together with their corresponding color. . . . . 81

7.1 The table shows the identified types of errors (rows) and the introduced methods (columns) for label transfer. It gives a qualitative overview of which method treats which type of error. The symbols in the table are as follows: "-" is not considered, "o" is treated implicitly and "+" is considered explicitly. . . . . 87

7.2 Results on the test set for all trials using different feature combinations. . . . . 94

7.3 Ablation study for the correction of label noise with scanstrip network (SNet). The indices indicate the used features. . . . . 96

7.4 Ablation study for finding a good input window size for network $\text{SNet}_{h,\xi,r,\delta,n}$. For better readability the indices are omitted. . . . . 97

7.5 The performance of SNet compared to other network architectures. Results marked by an asterisk indicate that the best performance on the test set was achieved due to early stopping with the second strategy . . . . . 98

7.6 The table shows the support for the training and test subsets. The first row shows the total amount of labelled pixels for each class. The following rows show the support for each subset in thousands (k) and percent of the total. The last set (leftover) shows the amount of data that is not used for training or testing because it is not associated with any 3D point. . . . 107

7.7 Table with all the results for the individual ablation studies. The mIoU is given for the validation (Val mIoU) and testing sets (Test mIoU). In addition, the test MIoU is also shown separately for static classes (third column) and dynamic classes (fourth column). . . . . 116

7.8 Comparison of methods applied directly to the scanstrip and LTNet. All networks were trained, validated and tested on the same sets. The input patch size is $256 \times 256$ in all cases. . . . . 124

8.1 FID and MS-SSIM scores computed for every campaign. The closer FID is to zero the better. MS-SSIM is bound between -1 and 1, where 1 indicates a perfect match between real and generated images. . . . . 128

8.2 IoU between real and synthesized images (Karlsruhe) and also between synthesized images and the reference set 6.1.1 (Reference Set). As Baseline the IoU is also given between the original MMS-images and the reference set (MMS-Set). Predictions were made with Deeplabv3+ pretrained on Cityscapes . . . . . 129

8.3 Comparison of achieved mIoU for all label transfer experiments. $\text{Deeplabv3+}_{3D}$ is displayed as reference when real MMS images are used as source for the label transfer. The others are using synthetic images with different pretrained DCNNs for semantic segmentation. . . . . 137

8.4 Quantitative Results for the proposed method, listed her as GAN. Numbers marked with † are taken from Stutz and Geiger (2018). Note that they use a slightly different voxel grid for the ShapeNet classes. . . . . 142

# Bibliography


Adams, R., Bischof, L., 1994. Seeded region growing. IEEE Transactions on Pattern Analysis & Machine Intelligence Vol. 16, 641–647.

Araujo, A., Norris, W., Sim, J., 2019. Computing receptive fields of convolutional neural networks. Distill Https://distill.pub/2019/computing-receptive-fields.

Atienza, R., 2019. A conditional generative adversarial network for rendering point clouds, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Computer Vision Foundation / IEEE. pp. 10–17.

Bahdanau, D., Cho, K., Bengio, Y., 2015. Neural machine translation by jointly learning to align and translate, in: Proceedings of the International Conference on Learning Representations (ICLR).

Barbosa, A., Marinho, T., Martin, N., Hovakimyan, N., 2020. Multi-stream cnn for spatial resource allocation: A crop management application, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, pp. 258–266.

Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., Gall, J., 2019. Semantickitti: A dataset for semantic scene understanding of lidar sequences, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 9296–9306.

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., 2006. Greedy layer-wise training of deep networks, in: Schölkopf, B., Platt, J.C., Hofmann, T. (Eds.), Advances in Neural Information Processing Systems 19, MIT Press. pp. 153–160.

Bentley, J.L., 1975. Multidimensional binary search trees used for associative searching. Communications of the ACM Vol. 18, 509–517.

Biasutti, P., Bugeau, A., Aujol, J., Brédif, M., 2019a. Riu-net: Embarrassingly simple semantic segmentation of 3d lidar point cloud. CoRR Vol. abs/1905.08748. `arXiv:1905.08748`.

Biasutti, P., Lepetit, V., Aujol, J., Bredif, M., Bugeau, A., 2019b. Lu-net: An efficient network for 3d lidar point cloud semantic segmentation based on end-to-end-learned 3d features and u-net, in: Proceedings of the IEEE International Conference on Computer Vision Workshop (ICCVW), IEEE Computer Society, Los Alamitos, CA, USA. pp. 942–950.

Boulch, A., Guerry, J., Le Saux, B., Audebert, N., 2018. Snapnet: 3d point cloud semantic labeling with 2d deep segmentation networks. Computers and Graphics Vol. 71, 189–198.

Boykov, Y., Funka-Lea, G., 2006. Graph cuts and efficient nd image segmentation. International Journal of Computer Vision Vol. 70, 109–131.

Breiman, L., 1997. Arcing the edge. Technical Report. Technical Report 486, Statistics Department, University of California at Berkeley.

Brenner, C., 2016. Scalable estimation of precision maps in a mapreduce framework, in: Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Association for Computing Machinery, New York, NY, USA. pp. 27:1–27:10.

Bresenham, J.E., 1965. Algorithm for computer control of a digital plotter. IBM Systems journal Vol. 4, 25–30.

Brown, R.A., 2015. Building a balanced $k$-d tree in $o(kn \log n)$ time. Journal of Computer Graphics Techniques (JCGT) Vol. 4, 50–68.

Cao, Z., Ma, L., Long, M., Wang, J., 2018. Partial adversarial domain adaptation, in: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (Eds.), Proceedings of the European Conference on Computer Vision (ECCV), Part VIII, Springer. pp. 139–155.

Cavegn, S., Haala, N., 2016. Image-based mobile mapping for 3d urban data capture. Photogrammetric Engineering & Remote Sensing Vol. 82, 925–933.

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

Chang, H., Han, J., Zhong, C., Snijders, A., Mao, J., 2018. Unsupervised transfer learning via multi-scale convolutional sparse coding for biomedical applications. IEEE Transactions on Pattern Analysis & Machine Intelligence Vol. 40, 1182–1194.

Chapelle, O., Schlkopf, B., Zien, A., 2010. Semi-Supervised Learning. 1st ed., The MIT Press.

Chechik, G., Sharma, V., Shalit, U., Bengio, S., 2010. Large scale online learning of image similarity through ranking. Journal of Machine Learning Research Vol. 11.

Chehata, N., Guo, L., Mallet, C., 2009. Airborne lidar feature selection for urban classification using random forests. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences Vol. 38, 207–212.

Chen, L., Papandreou, G., Schroff, F., Adam, H., 2017a. Rethinking atrous convolution for semantic image segmentation. CoRR Vol. abs/1706.05587. arXiv:1706.05587.

Chen, L., Zhu, Y., Papandreou, G., Schroff, F., Adam, H., 2018. Encoder-decoder with atrous separable convolution for semantic image segmentation, in: Proceedings of the European Conference on Computer Vision (ECCV), Part VII, Springer. pp. 833–851.

Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L., 2017b. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE Transactions on Pattern Analysis & Machine Intelligence Vol. 40, 834–848.

Chen, X., Chen, B., Mitra, N.J., 2020a. Unpaired point cloud completion on real scans using adversarial training, in: Proceedings of the International Conference on Learning Representations (ICLR).

Chen, Y.C., Tan, D.S., Cheng, W.H., Hua, K.L., 2019. 3d object completion via class-conditional generative adversarial network, in: International Conference on Multimedia Modeling, Springer. pp. 54–66.

Chen, Z., Zhang, R., Zhang, G., Ma, Z., Lei, T., 2020b. Digging into pseudo label: A low-budget approach for semi-supervised semantic segmentation. IEEE Access Vol. 8, 41830–41837.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E., 2014. cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759 .

Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 1800–1807.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B., 2016. The cityscapes dataset for semantic urban scene understanding, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 3213–3223.

Criminisi, A., Pérez, P., Toyama, K., 2004. Region filling and object removal by exemplar-based image inpainting. IEEE Transactions on Image Processing Vol. 13, 1200–1212.

Dai, A., Qi, C.R., Nießner, M., 2017. Shape completion using 3d-encoder-predictor cnns and shape synthesis, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Los Alamitos, CA, USA. pp. 6545–6554.

Demir, U., Ünal, G.B., 2018. Patch-based image inpainting with generative adversarial networks. CoRR Vol. abs/1803.07422. arXiv:1803.07422.

Do, T., Nguyen, H., Nguyen, T., Vu, H., Tran, T., Le, T., 2017. Plant identification using score-based fusion of multi-organ images, in: International Conference on Knowledge and Systems Engineering (KSE), pp. 191–196.

Doersch, C., Gupta, A., Efros, A.A., 2015a. Unsupervised visual representation learning by context prediction, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 1422–1430.

Doersch, C., Gupta, A., Efros, A.A., 2015b. Unsupervised visual representation learning by context prediction, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), IEEE Computer Society. pp. 1422–1430.

Dolata, P., Mrzygłód, M., Reiner, J., 2017. Double-stream convolutional neural networks for machine vision inspection of natural products. Applied Artifical Intelligence Vol. 31, 643–659.

Domingos, P., 2000. A unified bias-variance decomposition, in: Proceedings of International Conference on Machine Learning (ICML), pp. 231–238.

Dosovitskiy, A., Fischer, P., Springenberg, J.T., Riedmiller, M., Brox, T., 2015. Discriminative unsupervised feature learning with exemplar convolutional neural networks. IEEE Transactions on Pattern Analysis & Machine Intelligence Vol. 38, 1734–1747.

Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A., 2010. The pascal visual object classes (voc) challenge. International Journal of Computer Vision Vol. 88, 303–338.

Farabet, C., Couprie, C., Najman, L., LeCun, Y., 2012. Learning hierarchical features for scene labeling. IEEE Transactions on Pattern Analysis & Machine Intelligence Vol. 35, 1915–1929.

Fedkiw, R., Osher, S., Zhao, H., 2001. Fast surface reconstruction using the level set method, in: Proceedings of 1st IEEE Workshop on Variational and Level Set Methods in Computer Vision, IEEE Computer Society, Los Alamitos, CA, USA. p. 194.

Feng, Y., Zhang, Z., Zhao, X., Ji, R., Gao, Y., 2018. Gvcnn: Group-view convolutional neural networks for 3d shape recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 264–272.

Feng, Z., Zhou, Q., Cheng, G., Tan, X., Shi, J., Ma, L., 2020. Semi-supervised semantic segmentation via dynamic self-training and class-balanced curriculum. CoRR Vol. abs/2004.08514. arXiv:2004.08514.

Firman, M., Mac Aodha, O., Julier, S., Brostow, G.J., 2016. Structured prediction of unobserved voxels from a single depth image, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5431–5440.

Floros, G., Leibe, B., 2012. Joint 2d-3d temporally consistent semantic segmentation of street scenes, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE. pp. 2823–2830.

Förstner, W., Wrobel, B., 2016. Photogrammetric Computer Vision: Geometry, Orientation and Reconstruction. Geometry and Computing, Springer International Publishing.

Freund, Y., Schapire, R.E., 1996. Experiments with a new boosting algorithm, in: Saitta, L. (Ed.), Proceedings of International Conference on Machine Learning (ICML), Morgan Kaufmann. pp. 148–156.

Friedman, J.H., 2002. Stochastic gradient boosting. Computational statistics & data analysis Vol. 38, 367–378.

Geiger, A., Lenz, P., Stiller, C., Urtasun, R., 2013. Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research Vol. 32, 1231–1237.

Geman, S., Bienenstock, E., Doursat, R., 1992. Neural networks and the bias/variance dilemma. Neural computation Vol. 4, 1–58.

George, D., Shen, H., Huerta, E.A., 2017. Deep transfer learning: A new deep learning glitch classification method for advanced LIGO. CoRR Vol. abs/1706.07446. arXiv:1706.07446.

Geras, K.J., Wolfson, S., Kim, S.G., Moy, L., Cho, K., 2017. High-resolution breast cancer screening with multi-view deep convolutional neural networks. CoRR Vol. abs/1703.07047. arXiv:1703.07047.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press. http://www.deeplearningbook.org.

Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y., 2014. Generative adversarial nets, in: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems 27, pp. 2672–2680.

Grilli, E., Menna, F., Remondino, F., 2017. A review of point clouds segmentation and classification algorithms. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences Vol. 42, 339.

Han, X., Li, Z., Huang, H., Kalogerakis, E., Yu, Y., 2017. High-resolution shape completion using deep neural networks for global structure and local geometry inference, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), IEEE Computer Society, Los Alamitos, CA, USA. pp. 85–93.

Haralick, R.M., Shapiro, L.G., 1985. Image segmentation techniques. Computer Vision, Graphics, and Image Processing Vol. 29, 100–132.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 770–778.

Hermans, A., Floros, G., Leibe, B., 2014. Dense 3d semantic mapping of indoor scenes from rgb-d images, in: IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 2631–2638.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S., 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA. pp. 6629–6640.

Ho, T.K., 1995. Random decision forests, in: Proceedings of 3rd international conference on document analysis and recognition, IEEE. pp. 278–282.

Hoffman, J., Tzeng, E., Park, T., Zhu, J., Isola, P., Saenko, K., Efros, A.A., Darrell, T., 2018. Cycada: Cycle-consistent adversarial domain adaptation, in: Dy, J.G., Krause, A. (Eds.), Proceedings of International Conference on Machine Learning (ICML), PMLR. pp. 1994–2003.

Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural Networks Vol. 4, 251–257.

Huang, J., Li, J., Yu, D., Deng, L., Gong, Y., 2013. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers, in: IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP), IEEE. pp. 7304–7308.

Huang, Q., Wang, W., Neumann, U., 2018. Recurrent slice networks for 3d segmentation of point clouds, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 2626–2635.

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of International Conference on Machine Learning (ICML), JMLR.org. pp. 448–456.

Isola, P., Zhu, J., Zhou, T., Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 5967–5976.

Jing, L., Tian, Y., 2019. Self-supervised visual feature learning with deep neural networks: A survey. IEEE Transactions on Pattern Analysis & Machine Intelligence , 1–1.

Johnson, J., Alahi, A., Fei-Fei, L., 2016. Perceptual losses for real-time style transfer and super-resolution, in: Proceedings of the European Conference on Computer Vision (ECCV), Part II, Springer. pp. 694–711.

Kaufman, A., Cohen, D., Yagel, R., 1993. Volume graphics. Computer Vol. 26, 51–64.

Kingma, D.P., Ba, J., 2015. Adam: A method for stochastic optimization, in: Proceedings of the International Conference on Learning Representations (ICLR).

Kingma, D.P., Welling, M., 2014. Auto-encoding variational bayes URL: http://arxiv.org/abs/1312.6114.

Kosiorek, A.R., Bewley, A., Posner, I., 2017. Hierarchical attentive recurrent tracking, in: Advances in Neural Information Processing Systems 30, pp. 3053–3061.

Kraevoy, V., Sheffer, A., 2005. Template-based mesh completion, in: Third Eurographics Symposium on Geometry Processing, Eurographics Association. pp. 13–22.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, Curran Associates, Inc.. pp. 1106–1114.

Kumar, A., Singh, U.P., 2012. Image segmentation using graphical models: a survey. The International Journal of Emerging Technology and Advanced Engineering Vol. 2, 290–294.

Ladicky, L., Sturgess, P., Alahari, K., Russell, C., Torr, P.H.S., 2010. What, where and how many? combining object detectors and crfs, in: Proceedings of the European Conference on Computer Vision (ECCV), Part IV, Springer. pp. 424–437.

Laine, S., Karras, T., 2010. Efficient sparse voxel octrees. EEE Transactions on Visualization and Computer Graphics Vol. 17, 1048–1059.

Le, T., Bui, G., Duan, Y., 2017. A multi-view recurrent neural network for 3d mesh segmentation. Computers and Graphics Vol. 66, 103–112. Shape Modeling International 2017.

Le, T., Duan, Y., 2018. Pointgrid: A deep network for 3d shape understanding, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 9204–9214.

LeCun, Y., 1989. Generalization and network design strategies. Elsevier.

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D., 1989. Backpropagation applied to handwritten zip code recognition. Neural computation Vol. 1, 541–551.

Lee, D.H., et al., 2013. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks, in: Workshop on challenges in representation learning, ICML.

Lee, S.H., Chan, C.S., Remagnino, P., 2018. Multi-organ plant classification based on convolutional and recurrent neural networks. IEEE Transactions on Image Processing Vol. 27, 4287–4301.

Lerma, C.D.C., Kosecka, J., 2014. Semantic segmentation with heterogeneous sensor coverages, in: IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 2639–2645. doi:10.1109/ICRA.2014.6907237.

Li, J., Chen, B.M., Lee, G.H., 2018a. So-net: Self-organizing network for point cloud analysis, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 9397–9406.

Li, J., Yang, B., Wu, W., Dai, W., Chen, C., Zou, X., Tian, M., 2017. 3d mobile mapping with a low cost uav system. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences Vol. 42, 127.

Li, R., Li, X., Fu, C., Cohen-Or, D., Heng, P., 2019. PU-GAN: A point cloud upsampling adversarial network, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), IEEE. pp. 7202–7211.

Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B., 2018b. Pointcnn: Convolution on x-transformed points, in: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 31, pp. 828–838.

Li, Z., Zhang, L., Tong, X., Du, B., Wang, Y., Zhang, L., Zhang, Z., Liu, H., Mei, J., Xing, X., Mathiopoulos, P.T., 2016. A three-step approach for TLS point cloud classification. IEEE Transactions on Geoscience and Remote Sensing Vol. 54, 5412–5424.

Lim, E.H., Suter, D., 2009. 3d terrestrial lidar classifications with super-voxels and multi-scale conditional random fields. Computer-Aided Design Vol. 41, 701–710. Selected Papers from the 2007 New Advances in Shape Analysis and Geometric Modeling Workshop.

Lin, C., Kumar, A., 2018. Contactless and partial 3d fingerprint recognition using multi-view deep representation. Lecture Notes in Computer Science Vol. 83, 314–327.

Lin, J., 1991. Divergence measures based on the shannon entropy. IEEE Transactions on Information Theory Vol. 37, 145–151.

Lodha, S.K., Fitzpatrick, D.N., Helmbold, D.P., 2007. Aerial lidar data classification using adaboost, in: Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM), IEEE Computer Society. pp. 435–442.

Long, J., Shelhamer, E., Darrell, T., 2015. Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3431–3440.

Long, M., Zhu, H., Wang, J., Jordan, M.I., 2016. Unsupervised domain adaptation with residual transfer networks, in: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 29, pp. 136–144.

Lu, C., Dubbelman, G., 2020. Learning to complete partial observations from unpaired prior knowledge. Lecture Notes in Computer Science Vol. 107, 107426.

Luo, Z., Zou, Y., Hoffman, J., Li, F., 2017. Label efficient learning of transferable representations acrosss domains and tasks, in: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30, pp. 165–177.

Luong, T., Pham, H., Manning, C.D., 2015. Effective approaches to attention-based neural machine translation, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP), The Association for Computational Linguistics. pp. 1412–1421.

Ma, L., Stückler, J., Kerl, C., Cremers, D., 2017. Multi-view deep learning for consistent semantic mapping with rgb-d cameras, in: International Conference on Intelligent Robots and Systems (IROS), IEEE. pp. 598–605.

Mallet, C., Soergel, U., Bretar, F., 2008. Analysis of full-waveform lidar data for classification of urban areas. PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science Vol. 5.

Mao, X., Li, Q., Xie, H., Lau, R.Y.K., Wang, Z., Smolley, S.P., 2017. Least squares generative adversarial networks, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), IEEE Computer Society. pp. 2813–2821.

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

Meagher, D., 1982. Geometric modeling using octree encoding. Computer graphics and image processing Vol. 19, 129–147.

Milioto, A., Vizzo, I., Behley, J., Stachniss, C., 2019. Rangenet ++: Fast and accurate lidar semantic segmentation, in: International Conference on Intelligent Robots and Systems (IROS), IEEE. pp. 4213–4220.

Mirza, M., Osindero, S., 2014. Conditional generative adversarial nets. CoRR Vol. abs/1411.1784. arXiv:1411.1784.

Mitra, N.J., Guibas, L.J., Pauly, M., 2006. Partial and approximate symmetry detection for 3d geometry. ACM Transactions on Graphics Vol. 25, 560–568.

Mitra, N.J., Pauly, M., Wand, M., Ceylan, D., 2013. Symmetry in 3d geometry: Extraction and applications. Computer Graphics Forum Vol. 32, 1–23.

Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y., 2018. Spectral normalization for generative adversarial networks, in: Proceedings of the International Conference on Learning Representations (ICLR), OpenReview.net.

Murphy, K.P., 2012. Machine learning - a probabilistic perspective. Adaptive computation and machine learning series, MIT Press.

Nassar, A.S., Lefevre, S., Wegner, J.D., 2019. Simultaneous multi-view instance detection with learned geometric soft-constraints, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 6558–6567.

Nguyen, A., Le, B., 2013. 3d point cloud segmentation: A survey, in: IEEE Conference on Robotics, Automation and Mechatronics (RAM), IEEE. pp. 225–230.

Niemeyer, J., Rottensteiner, F., Soergel, U., 2012. Conditional random fields for lidar point cloud classification in complex urban areas. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences Vol. 1, 263–268.

Niemeyer, J., Rottensteiner, F., Soergel, U., 2014. Contextual classification of lidar data and building object detection in urban areas. ISPRS Journal of Photogrammetry and Remote Sensing Vol. 87, 152–165.

Noh, H., Hong, S., Han, B., 2015. Learning deconvolution network for semantic segmentation, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 1520–1528.

Noroozi, M., Favaro, P., 2016. Unsupervised learning of visual representations by solving jigsaw puzzles, in: Leibe, B., Matas, J., Sebe, N., Welling, M. (Eds.), Proceedings of the European Conference on Computer Vision (ECCV), Part VI, Springer. pp. 69–84.

Odena, A., Olah, C., Shlens, J., 2017. Conditional image synthesis with auxiliary classifier gans, in: International Conference on Machine Learning (ICML), PMLR. pp. 2642–2651.

Papert, M.M.S., Minsky, M., 1969. Perceptrons: an introduction to computational geometry. MIT Press.

Pauly, M., Mitra, N.J., Giesen, J., Gross, M.H., Guibas, L.J., 2005. Example-based 3d scan completion, in: Third Eurographics Symposium on Geometry Processing, Vienna, Austria, Eurographics Association. pp. 23–32.

Pauly, M., Mitra, N.J., Wallner, J., Pottmann, H., Guibas, L.J., 2008. Discovering structural regularity in 3d geometry. ACM Transactions on Graphics Vol. 27, 43.

Peters, T., Brenner, C., 2019. Automatic generation of large point cloud training datasets using label transfer, in: Publikationen der DGPF, Band 28, pp. 68–82.

Peters, T., Brenner, C., 2020. Conditional adversarial networks for multimodal photo-realistic point cloud rendering. PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science Vol. 88, 257–269.

Peters, T., Brenner, C., Song, M., 2020. Improving deep learning based semantic segmentation with multi view outlier correction. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences Vol. 43, 711–716.

Podolak, J., Shilane, P., Golovinskiy, A., Rusinkiewicz, S., Funkhouser, T., 2006. A planar-reflective symmetry transform for 3d shapes. ACM Transactions on Graphics Vol. 25, 549–559.

Puente, I., González-Jorge, H., Martínez-Sánchez, J., Arias, P., 2013. Review of mobile mapping and surveying technologies. Measurement Vol. 46, 2127–2145.

Qi, C.R., Su, H., Mo, K., Guibas, L.J., 2017a. Pointnet: Deep learning on point sets for 3d classification and segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 77–85.

Qi, C.R., Su, H., Mo, K., Guibas, L.J., 2017b. Pointnet: Deep learning on point sets for 3d classification and segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 652–660.

Qi, C.R., Yi, L., Su, H., Guibas, L.J., 2017c. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, in: Advances in Neural Information Processing Systems, Curran Associates, Inc.. pp. 5099–5108.

Qiu, Z., Yan, F., Zhuang, Y., Leung, H., 2019. Outdoor semantic segmentation for ugvs based on cnn and fully connected crfs. IEEE Sensors Journal Vol. 19, 4290–4298.

Quinlan, J.R., 1986. Induction of decision trees. Machine learning Vol. 1, 81–106.

Rabbani, T., Van Den Heuvel, F., Vosselmann, G., 2006. Segmentation of point clouds using smoothness constraint. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences Vol. 36, 248–253.

Recht, B., Roelofs, R., Schmidt, L., Shankar, V., 2018. Do CIFAR-10 classifiers generalize to cifar-10? CoRR Vol. abs/1806.00451. `arXiv:1806.00451`.

Recht, B., Roelofs, R., Schmidt, L., Shankar, V., 2019. Do ImageNet classifiers generalize to ImageNet?, in: Proceedings of International Conference on Machine Learning (ICML), PMLR. pp. 5389–5400.

Rezende, D.J., Eslami, S.A., Mohamed, S., Battaglia, P., Jaderberg, M., Heess, N., 2016. Unsupervised learning of 3d structure from images, in: Advances in Neural Information Processing Systems, pp. 4996–5004.

Riegler, G., Osman Ulusoy, A., Geiger, A., 2017a. Octnet: Learning deep 3d representations at high resolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3577–3586.

Riegler, G., Ulusoy, A.O., Bischof, H., Geiger, A., 2017b. Octnetfusion: Learning depth fusion from data, in: Proceedings of the International Conference on 3D Vision (3DV), pp. 57–66.

Rifai, S., Mesnil, G., Vincent, P., Muller, X., Bengio, Y., Dauphin, Y., Glorot, X., 2011. Higher order contractive auto-encoder, in: Joint European conference on machine learning and knowledge discovery in databases, Springer. pp. 645–660.

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer. pp. 234–241.

Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review Vol. 65, 386.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning representations by back-propagating errors. Nature Vol. 323, 533–536.

Rusu, R.B., Cousins, S., 2011. 3d is here: Point cloud library (pcl), in: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China.

Salehi, S.S.M., Erdogmus, D., Gholipour, A., 2017. Tversky loss function for image segmentation using 3d fully convolutional deep networks, in: Machine Learning in Medical Imaging, Springer International Publishing, Cham. pp. 379–387.

Sandler, M., Howard, A.G., Zhu, M., Zhmoginov, A., Chen, L., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 4510–4520.

Schachtschneider, J., Brenner, C., 2020. Creating multi-temporal maps of urban environments for improved localization of autonomous vehicles. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences Vol. 43, 317–323.

Schmidt, A., Rottensteiner, F., Soergel, U., 2012. Classification of airborne laser scanning data in wadden sea areas using conditional random fields. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences Vol. XXXIX-B3, 161–166.

Schön, S., Brenner, C., Alkhatib, H., Coenen, M., Dbouk, H., Garcia-Fernandez, N., Fischer, C., Heipke, C., Lohmann, K., Neumann, I., Nguyen, U., Paffenholz, J.A., Peters, T., Rottensteiner, F., Schachtschneider, J., Sester, M., Sun, L., Vogel, S., Voges, R., Wagner, B., 2018. Integrity and collaboration in dynamic sensor networks. Sensors Vol. 18.

Scudder, H., 1965. Probability of error of some adaptive pattern-recognition machines. IEEE Transactions on Information Theory Vol. 11, 363–371.

Seeland, M., Mäder, P., 2021. Multi-view classification with convolutional neural networks. PLOS ONE Vol. 16, e0245230.

Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., Levine, S., 2018. Time-contrastive networks: Self-supervised learning from video, in: IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 1134–1141.

Setio, A.A.A., Ciompi, F., Litjens, G., Gerke, P., Jacobs, C., van Riel, S.J., Wille, M.M.W., Naqibullah, M., Sánchez, C.I., van Ginneken, B., 2016. Pulmonary nodule detection in ct images: False positive reduction using multi-view convolutional networks. IEEE Transactions on Medical Imaging Vol. 35, 1160–1169.

Sharma, A., Grau, O., Fritz, M., 2016. Vconv-dae: Deep volumetric shape learning without object labels, in: Proceedings of the European Conference on Computer Vision (ECCV) Workshops, Springer. pp. 236–250.

Shen, J., Hao, X., Liang, Z., Liu, Y., Wang, W., Shao, L., 2016. Real-time superpixel segmentation by dbscan clustering algorithm. IEEE Transactions on Image Processing Vol. 25, 5933–5942.

Shih, F.Y., Cheng, S., 2005. Automatic seeded region growing for color image segmentation. Image and Vision Computing Vol. 23, 877–886.

Shmelkov, K., Schmid, C., Alahari, K., 2018. How good is my gan?, in: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (Eds.), Proceedings of the European Conference on Computer Vision (ECCV), Part II, Springer. pp. 218–234.

Simonyan, K., Zisserman, A., 2015. Very deep convolutional networks for large-scale image recognition, in: Proceedings of the International Conference on Learning Representations (ICLR).

Sipiran, I., Gregor, R., Schreck, T., 2014. Approximate symmetry detection in partial 3d meshes. Computer Graphics Forum Vol. 33, 131–140.

Stutz, D., Geiger, A., 2018. Learning 3d shape completion from laser scan data with weak supervision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Los Alamitos, CA, USA. pp. 1955–1964.

Su, H., Maji, S., Kalogerakis, E., Learned-Miller, E., 2015. Multi-view convolutional neural networks for 3d shape recognition, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 945–953.

Sudre, C.H., Li, W., Vercauteren, T., Ourselin, S., Cardoso, M.J., 2017. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations, in: Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support - Third International Workshop, (DLMIA), and 7th International Workshop, (ML-CDS), Held in Conjunction with (MICCAI), Springer. pp. 240–248.

Sun, K., Xiao, B., Liu, D., Wang, J., 2019. Deep high-resolution representation learning for human pose estimation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Computer Vision Foundation / IEEE. pp. 5693–5703.

Sung, M., Kim, V.G., Angst, R., Guibas, L., 2015. Data-driven structural priors for shape completion. ACM Transactions on Graphics Vol. 34, 175.

Sutskever, I., Vinyals, O., Le, Q.V., 2014. Sequence to sequence learning with neural networks, in: Advances in Neural Information Processing Systems 27, pp. 3104–3112.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.E., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 1–9.

Tchapmi, L.P., Choy, C.B., Armeni, I., Gwak, J., Savarese, S., 2017. Segcloud: Semantic segmentation of 3d point clouds, in: Proceedings of the International Conference on 3D Vision (3DV), IEEE Computer Society. pp. 537–547.

Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J., 2019. Kpconv: Flexible and deformable convolution for point clouds, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 6411–6420.

Tzeng, E., Hoffman, J., Saenko, K., Darrell, T., 2017. Adversarial discriminative domain adaptation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 2962–2971.

Van Engelen, J.E., Hoos, H.H., 2020. A survey on semi-supervised learning. Machine Learning Vol. 109, 373–440.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017. Attention is all you need, in: Advances in Neural Information Processing Systems 30, pp. 5998–6008.

Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P., 2008. Extracting and composing robust features with denoising autoencoders, in: Cohen, W.W., McCallum, A., Roweis, S.T. (Eds.), Proceedings of International Conference on Machine Learning (ICML), ACM. pp. 1096–1103.

Vondrick, C., Shrivastava, A., Fathi, A., Guadarrama, S., Murphy, K., 2018. Tracking emerges by colorizing videos, in: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (Eds.), Proceedings of the European Conference on Computer Vision (ECCV), Part XIII, Springer. pp. 402–419.

Vosselman, G., Coenen, M., Rottensteiner, F., 2017. Contextual segment-based classification of airborne laser scanner data. ISPRS Journal of Photogrammetry and Remote Sensing Vol. 128, 354–371.

Vosselman, G., Maas, H. (Eds.), 2010. Airborne and terrestrial laser scanning. CRC Press, United Kingdom.

Wang, A., Cai, J., Lu, J., Cham, T., 2015. Mmss: Multi-modal sharable and specific feature learning for rgb-d object recognition, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 1125–1133.

Wang, M., Deng, W., 2018. Deep visual domain adaptation: A survey. Neurocomputing Vol. 312, 135–153.

Wang, T., Liu, M., Zhu, J., Tao, A., Kautz, J., Catanzaro, B., 2018a. High-resolution image synthesis and semantic manipulation with conditional gans, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 8798–8807.

Wang, Y., Chao, W.L., Garg, D., Hariharan, B., Campbell, M., Weinberger, K.Q., 2019. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8445–8453.

Wang, Y., Shi, T., Yun, P., Tai, L., Liu, M., 2018b. Pointseg: Real-time semantic segmentation based on 3d lidar point cloud. CoRR Vol. abs/1807.06288. `arXiv:1807.06288`.

Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P., 2004. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing Vol. 13, 600–612.

Wang, Z., Simoncelli, E., Bovik, A., 2003. Multiscale structural similarity for image quality assessment, pp. 1398–1402 Vol.2.

Wang, Z., Zhang, L., Fang, T., Mathiopoulos, P.T., Tong, X., Qu, H., Xiao, Z., Li, F., Chen, D., 2015. A multiscale and hierarchical feature extraction method for terrestrial laser scanning point cloud classification. IEEE Transactions on Geoscience and Remote Sensing Vol. 53, 2409–2425.

Weinmann, M., Jutzi, B., Hinz, S., Mallet, C., 2015. Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. ISPRS Journal of Photogrammetry and Remote Sensing Vol. 105, 286–304.

Wu, B., Wan, A., Yue, X., Keutzer, K., 2018. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud, in: IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 1887–1893.

Wu, B., Zhou, X., Zhao, S., Yue, X., Keutzer, K., 2019. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud, in: IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 4376–4382.

Wu, H., Prasad, S., 2017. Semi-supervised deep learning using pseudo labels for hyperspectral image classification. IEEE Transactions on Image Processing Vol. 27, 1259–1270.

Wu, J., Zhang, C., Xue, T., Freeman, B., Tenenbaum, J., 2016. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling, in: Advances in Neural Information Processing Systems 29, pp. 82–90.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J., 2015. 3d shapenets: A deep representation for volumetric shapes, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1912–1920.

Xiao, J., Quan, L., 2009. Multiple view semantic segmentation for street view images, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), IEEE. pp. 686–693.

Xie, J., Kiefel, M., Sun, M.T., Geiger, A., 2016. Semantic instance annotation of street scenes by 3d to 2d label transfer, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3688–3697.

Xie, Q., Luong, M., Hovy, E.H., Le, Q.V., 2020a. Self-training with noisy student improves imagenet classification, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE. pp. 10684–10695.

Xie, Y., Tian, J., Zhu, X.X., 2020b. Linking points with labels in 3d: A review of point cloud semantic segmentation. IEEE Geoscience and Remote Sensing Magazine Vol. 8, 38–59.

Yang, C., Lu, X., Lin, Z., Shechtman, E., Wang, O., Li, H., 2017. High-resolution image inpainting using multi-scale neural patch synthesis, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Los Alamitos, CA, USA. pp. 4076–4084.

Yang, Y., Feng, C., Shen, Y., Tian, D., 2018. Foldingnet: Point cloud auto-encoder via deep grid deformation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 206–215.

Yao, Y., Xu, K., Murasaki, K., Ando, S., Sagata, A., 2020. Pseudo-labelling-aided semantic segmentation on sparsely annotated 3d point clouds. IPSJ Transactions on Computer Vision and Applications Vol. 12, 1–13.

Yeh, R.A., Chen, C., Lim, T., Schwing, A.G., Hasegawa-Johnson, M., Do, M.N., 2017. Semantic image inpainting with deep generative models, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 6882–6890.

Yosinski, J., Clune, J., Bengio, Y., Lipson, H., 2014. How transferable are features in deep neural networks?, in: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems 27, pp. 3320–3328.

Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., Huang, T.S., 2018. Generative image inpainting with contextual attention, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society. pp. 5505–5514.

Yuan, W., Khot, T., Held, D., Mertz, C., Hebert, M., 2018. Pcn: Point completion network, in: Proceedings of the International Conference on 3D Vision (3DV), IEEE Computer Society, Los Alamitos, CA, USA. pp. 728–737.

Zeiler, M.D., Fergus, R., 2014. Visualizing and understanding convolutional networks, in: Fleet, D.J., Pajdla, T., Schiele, B., Tuytelaars, T. (Eds.), Proceedings of the European Conference on Computer Vision (ECCV), Part I, Springer. pp. 818–833.

Zhang, J., Lin, X., Ning, X., 2013. Svm-based classification of segmented airborne lidar point clouds in urban areas. Remote Sensing Vol. 5, 3749–3775.

Zhang, J., Zhan, R., Sun, D., Pan, G., 2019a. Symmetry-Aware Face Completion with Generative Adversarial Networks. volume Vol. 11364. pp. 289–304.

Zhang, J., Zhao, X., Chen, Z., Lu, Z., 2019b. A review of deep learning-based semantic segmentation for point cloud. IEEE Access Vol. 7, 179118–179133.

Zhang, R., Li, G., Li, M., Wang, L., 2018. Fusion of images and point clouds for the semantic segmentation of large-scale 3d scenes based on deep learning. ISPRS Journal of Photogrammetry and Remote Sensing Vol. 143, 85–96.

Zhou, X., Karpur, A., Gan, C., Luo, L., Huang, Q., 2018. Unsupervised domain adaptation for 3d keypoint estimation via view consistency, in: Proceedings of the European Conference on Computer Vision (ECCV), pp. 137–153.

Zhu, H., Long, M., Wang, J., Cao, Y., 2016. Deep hashing network for efficient similarity retrieval, in: Schuurmans, D., Wellman, M.P. (Eds.), Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI Press. pp. 2415–2421.

Zhu, J., Zhang, R., Pathak, D., Darrell, T., Efros, A.A., Wang, O., Shechtman, E., 2017. Toward multimodal image-to-image translation, in: Advances in Neural Information Processing Systems 30, pp. 465–476.

Zhu, Y., Zhang, Z., Wu, C., Zhang, Z., He, T., Zhang, H., Manmatha, R., Li, M., Smola, A.J., 2020. Improving semantic segmentation via self-training. CoRR Vol. abs/2004.14960. `arXiv:2004.14960`.

Zoph, B., Ghiasi, G., Lin, T., Cui, Y., Liu, H., Cubuk, E.D., Le, Q., 2020. Rethinking pre-training and self-training, in: Advances in Neural Information Processing Systems 33.

Zou, Y., Yu, Z., Kumar, B.V.K.V., Wang, J., 2018. Unsupervised domain adaptation for semantic segmentation via class-balanced self-training, in: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (Eds.), Proceedings of the European Conference on Computer Vision (ECCV), Part III, Springer. pp. 297–313.

Zou, Y., Zhang, Z., Zhang, H., Li, C., Bian, X., Huang, J., Pfister, T., 2020. Pseudoseg: Designing pseudo labels for semantic segmentation. CoRR Vol. abs/2010.09713. `arXiv:2010.09713`.

Zwicker, M., Pfister, H., Van Baar, J., Gross, M., 2001. Surface splatting, in: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, ACM. pp. 371–378.

## Resume

### Personal Details

Torben Peters

Born in Langenhagen, Germany on 07.03.1988

### Education

**1994 - 1998**: Gebrüder Grimm Schule Letter

**1998 - 2000**: Geschwister Scholl Schule Letter

**2000 - 2008**: Georg Büchner Gymnasium Letter, Abitur

**10.2009 - 03.2014**: University of Lübeck, B.Sc. Computer Science and Robotics

**10.2014 - 04.2017**: Leibniz University Hanover, M.Sc. Navigation and Field Robotics

### Occupation

**05.2017-04.2021**: Researcher at Institute of Cartography and Geoinformatics, Leibniz University Hanover, Germany

## Acknowledgements

In the last part of the thesis I would like to express that I could never have written this work without the help of all my wonderful friends, relatives and colleagues. Or to sum it up with the Beatles: I only made it this far “with a little help from my friends”.

First of all, I would like to thank my supervisor, Prof. Claus Brenner, who taught me in my master’s program, supervised my master’s thesis and then mentored me at the IKG. I would like to thank him for steering my career in the current direction. I have learned a lot from him and he gave me the freedom to follow my ideas and was always interested and encouraging. I would also like to thank him for guiding me through my research and dissertation which helped me a lot.

I would also like to thank Prof. Franz Rottensteiner and Prof. Konrad Schindler for reviewing my dissertation. In addition, I would like to thank Prof. Rottensteiner for the many suggestions he gave me during the i.c.sens program. And a special thanks to Prof. Konrad Schindler for supervising me at the ETH Zurich. Even within the short time at his institute I could learn a lot.

I have always felt at home at the IKG and that has a lot to do with my great colleagues and my boss Prof. Monika Sester. Thank you for all the nice conversations and the help you have given me. I wish you all the best in your research and your future path. A very special thank you goes to my colleagues and friends Dennis Wittich and Bashir Kazimi, you are the best! I want to thank you for all your help and for always being there for me.

Finally I want to acknowledge that my work was funded by the German Research Foundation (DFG) as a part of the Research Training Group GRK2159, ‘Integrity and collaboration in dynamic sensor networks (i.c.sens)’.

Die letzten Grüße sind auf Deutsch, weil ich mich damit direkt an meine Familie wenden möchte. Vielen Dank an meine Eltern (Ute und Detlef), an meine Schwester (Anne-Stine) und an meinen Großvater (Jobst) - Danke, dass ihr immer für mich da seid und dass es euch gibt, ich liebe euch! Ganz besonders möchte ich meiner Freundin Olga danken. Sie trägt einen großen Teil dieser Arbeit mit. Wahrscheinlich wäre ich ohne sie, ohne ihre Unterstützung und Liebe nicht unbeschadet durch diese Zeit gekommen. - Ja tebja ljublju, Olečka!

# Wissenschaftliche Arbeiten der Fachrichtung Geodäsie und Geoinformatik der Leibniz Universität Hannover

*(Eine vollständige Liste der Wiss. Arb. ist beim Geodätischen Institut, Nienburger Str. 1, 30167 Hannover erhältlich.)*

| | | |
|---|---|---|
| Nr. 353 | ZHAO, Xin: | Terrestrial Laser Scanning Data Analysis for Deformation Monitoring (Diss. 2019) |
| Nr. 354 | HAGHIGHI, Mahmud Haghshenas: | Local and Large Scale InSAR Measurement of Ground Surface Deformation (Diss. 2019) |
| Nr. 355 | BUREICK, Johannes: | Robuste Approximation von Laserscan-Profilen mit B-Spline-Kurven (Diss. 2020) |
| Nr. 356 | BLOTT, Gregor: | Multi-View Person Re-Identification (Diss. 2020) |
| Nr. 357 | MAAS, Alina Elisabeth: | Klassifikation multitemporaler Fernerkundungsdaten unter Verwendung fehlerbehafteter topographischer Daten (Diss. 2020) |
| Nr. 358 | NGUYEN, Uyen: | 3D Pedestrian Tracking Using Neighbourhood Constraints (Diss. 2020) |
| Nr. 359 | KIELER, Birgit: | Schema-Matching in räumlichen Datensätzen durch Zuordnung von Objektinstanzen (Diss. 2020) |
| Nr. 360 | PAUL, Andreas: | Domänenadaption zur Klassifikation von Luftbildern (Diss. 2020) |
| Nr. 361 | UNGER, Jakob: | Integrated Estimation of UAV Image Orientation with a Generalised Building Model (Diss. 2020) |
| Nr. 362 | COENEN, Max: | Probabilistic Pose Estimation and 3D Reconstruction of Vehicles from Stereo Images (Diss. 2020) |
| Nr. 363 | GARCIA FERNANDEZ, Nicolas: | Simulation Framework for Collaborative Navigation: Development - Analysis – Optimization (Diss. 2020) |
| Nr. 364 | VOGEL, Sören: | Kalman Filtering with State Constraints Applied to Multi-sensor Systems and Georeferencing (Diss. 2020) |
| Nr. 365 | BOSTELMANN, Jonas: | Systematische Bündelausgleichung großer photogrammetrischer Blöcke einer Zeilenkamera am Beispiel der HRSC-Daten (Diss. 2020) |
| Nr. 366 | OMIDALIZARANDI, Mohammad: | Robust Deformation Monitoring of Bridge Structures Using MEMS Accelero-meters and Image-Assisted Total Stations (Diss. 2020) |
| Nr. 367 | ALKHATIB, Hamza: | Fortgeschrittene Methoden und Algorithmen für die computergestützte geodätische Datenanalyse (Habil. 2020) |
| Nr. 368 | DARUGNA, Francesco: | Improving Smartphone-Based GNSS Positioning Using State Space Augmentation Techniques (Diss. 2021) |
| Nr. 369 | CHEN, Lin: | Deep learning for feature based image matching (Diss. 2021) |
| Nr. 370 | DBOUK, Hani: | Alternative Integrity Measures Based on Interval Analysis and Set Theory (Diss. 2021) |
| Nr. 371 | CHENG, Hao: | Deep Learning of User Behavior in Shared Spaces (Diss. 2021) |
| Nr. 372 | MUNDT Reinhard Walter: | Schätzung von Boden- und Gebäudewertanteilen aus Kaufpreisen bebauter Grundstücke (Diss. 2021) |
| Nr. 373 | WANG, Xin: | Robust and Fast Global Image Orientation (Diss. 2021) |
| Nr. 374 | REN, Le: | GPS-based Precise Absolute and Relative Kinematic Orbit Determination of Swarm Satellites under Challenging Ionospheric Conditions (Diss. 2021) |
| Nr. 375 | XU, Wei: | Automatic Calibration of Finite Element Analysis Based on Geometric Boundary Models from Terrestrial Laser Scanning (Diss. 2021) |
| Nr. 376 | FENG, Yu: | Extraction of Flood and Precipitation Observations from opportunistic Volunteered Geographic Information (Diss. 2021) |
| Nr. 377 | YANG, Chun: | A hierarchical deep learning framework for the verification of geospatial databases (Diss. 2021) |
| Nr. 378 | MEHLTRETTER, Max: | Uncertainty Estimation for Dense Stereo Matching using Bayesian Deep Learning (Diss. 2021) |
| Nr. 379 | KAZIMI, Bashir: | Self Supervised Learning for Detection of Archaeological Monuments in LiDAR Data (Diss. 2021) |
| Nr. 380 | PETERS, Torben: | Learning Multi-View 2D to 3D Label Transfer for Semi-Supervised Semantic Segmentation of Point Clouds (Diss. 2022) |

*Die Arbeiten werden im Rahmen des wissenschaftlichen Schriftenaustausches verteilt und sind nicht im Buchhandel erhältlich. Der Erwerb ist zu einem Stückpreis von € 25,00 bei den herausgebenden Instituten möglich.*