HAL
open science

2D-3D scene understanding for autonomous driving

Maximilian Jaritz

To cite this version:

Maximilian Jaritz. 2D-3D scene understanding for autonomous driving. Machine Learning [cs.LG]. Université Paris sciences et lettres, 2020. English. NNT : 2020UPSLM007 . tel-02921424

HAL Id: tel-02921424

https://pastel.hal.science/tel-02921424v1

Submitted on 25 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à MINES ParisTech

2D-3D Scene Understanding for Autonomous Driving

Compréhension 2D-3D de scènes pour la conduite autonome

Soutenue par

Maximilian JARITZ

Le 26 juin 2020

École doctorale n°621

ISMME Ingénierie des Systèmes, Matériaux, Mécanique, Énergétique

Spécialité

Informatique temps réel, robotique et automatique

Composition du jury :

Vincent LEPETIT Prof., École des Ponts ParisTech	*Président*
Gabriel BROSTOW Prof., University College London	*Rapporteur*
Frédéric JURIE Prof., Université de Caen Normandie	*Examinateur*
Angela DAI Dr., Technical University of Munich	*Examinateur*
Fawzi NASHASHIBI Prof., Inria	*Directeur de thèse*
Raoul DE CHARETTE Dr., Inria	*Examinateur*
Émilie WIRBEL Dr., Valeo	*Examinateur*

MINES ParisTech | PSL

Acknowledgements

First, I'd like to thank my mentors Raoul de Charette and Émilie Wirbel for having dedicated so much time advising me scientifically and on all other matters that are often overseen. I appreciate so much that you were always available for me and pushed me further. This thesis would not have been possible without your support.

Furthermore, I am very grateful to Patrick Pérez for having welcomed me to the excellent team of Valeo.ai, which he leads with great care, and for having taken the time to share his scientific insights in our meetings and bringing clarity to my writing. I would also like to thank Fawzi Nashashibi for setting up the framework for my thesis in his RITS team at Inria and his great handling of academic-industrial collaboration, as well as his insights into the field of robotics.

I also want to thank all the great people that I had the chance to work with. Etienne Perot, who introduced me to Deep Learning at Valeo, before starting this thesis. My managers at the Valeo Driving Assistance Research team: Julien Rebut, Vanessa Picron, Xavier Perrotton and Antoine Lafay, who always believed in me and made so much possible. Hao Su, who received me at his lab at UC San Diego where I worked with Jiayuan Gu who writes such good code. And Tuan-Hung Vu from Valeo.ai, who it was such a pleasure to work with and who introduced me to new research fields.

Moreover, I would like to thank Gabriel Brostow and Vincent Lepetit for carefully reviewing my thesis, as well as Frédéric Jurie and Angela Dai for evaluating me remotely in those times of COVID-19, and enabling me to see the context in which my work is placed.

I would also like to thank all the colleagues and lab mates for the discussions and happy moments that we shared: Marin Toromanoff, Thibault Buhet, Raphaël Rose-Andrieux, Fabio Pizzati, Luís Roldao, Anne Verroust, Jean-Marc Lasgouttes, Anne Mathurin, Ronald Yu, Rui Chen, Songfang Han, Matthieu Cord, Renaud Marlet, Alexandre Boulch, Spyros Gidaris, Gabriel de Marmiesse, Andrei Bursuc, Hedi Ben-younes, Himalaya Jain and so many other nice colleagues at Valeo Driving Assistance Research, Valeo.ai, Inria and UC San Diego.

Finally, I would like to thank my wife Anna, who was introduced to the ups and downs of academic work and always supported and grounded me, as well as my parents for their unconditional love.

Contents

1 Introduction **3**
1.1 Goals . . . 5
1.2 Autonomous driving context . . . 7
1.3 Thesis Structure . . . 9

2 End-to-End Driving with Deep Reinforcement Learning **13**
2.1 Introduction . . . 15
2.2 Reinforcement learning background . . . 18
2.3 Related Work . . . 20
2.4 Method . . . 23
2.4.1 Reinforcement learning framework . . . 24
2.4.2 Learning strategy . . . 27
2.5 Experiments . . . 31
2.5.1 Training setup . . . 31
2.5.2 Metrics . . . 31
2.5.3 Performance evaluation . . . 32
2.5.4 Ablation studies . . . 34
2.5.5 Generalization . . . 37
2.6 Discussion . . . 39
2.7 Conclusion . . . 40

3 Fusing sparse depth and dense RGB for depth completion **45**
3.1 Introduction . . . 46
3.2 Related Work . . . 49
3.3 Method . . . 54
3.3.1 Network Architecture . . . 54
3.3.2 Sparse Data Training . . . 56
3.3.3 Analysis of Validity Mask . . . 58
3.4 Experiments . . . 59
3.4.1 Datasets . . . 59
3.4.2 Implementation . . . 60
3.4.3 Depth completion . . . 60
3.4.4 Semantic Segmentation . . . 65
3.5 Discussion . . . 69
3.6 Conclusion . . . 72

4 Multi-view PointNet for 3D scene understanding 75
4.1 Introduction . . . 77
4.1.1 Outdoor vs. indoor data . . . 79
4.2 Related Work . . . 81
4.3 MVPNet . . . 85
4.3.1 Overview . . . 85
4.3.2 View Selection . . . 86
4.3.3 2D Encoder-Decoder Network . . . 87
4.3.4 2D-3D Feature Lifting Module . . . 87
4.3.5 3D Fusion Network . . . 88
4.4 Experiments on ScanNet . . . 89
4.4.1 ScanNet Dataset . . . 89
4.4.2 Implementation Details . . . 89
4.4.3 3D Semantic Segmentation Benchmark . . . 90
4.4.4 Robustness to Varying Point Cloud Density . . . 93
4.5 Ablation Studies . . . 93
4.5.1 Number of Views . . . 95
4.5.2 Feature Aggregation Module . . . 95
4.5.3 Fusion . . . 97
4.5.4 Stronger Backbone . . . 97
4.6 Experiments on S3DIS . . . 98
4.6.1 S3DIS Dataset . . . 98
4.6.2 3D Semantic Segmentation . . . 98
4.7 Conclusion . . . 99

5 2D-3D Cross-modal Unsupervised Domain Adaptation 103
5.1 Introduction . . . 104
5.2 Related Work . . . 106
5.3 xMUDA . . . 110
5.3.1 Architecture . . . 111
5.3.2 Learning Scheme . . . 112
5.4 Experiments . . . 114
5.4.1 Datasets . . . 114
5.4.2 Implementation Details . . . 115
5.4.3 Main Experiments . . . 116
5.4.4 Extension to Fusion . . . 118
5.5 Ablation Studies . . . 121
5.5.1 Segmentation Heads . . . 121
5.5.2 Cross-modal Learning on Source . . . 122
5.5.3 Cross-modal Learning for Oracle Training . . . 122
5.6 Conclusion . . . 123

6 Conclusion **127**
6.1 Contributions 127
6.2 Future Work 128

Publications **131**

Bibliography **133**

A Fusing sparse depth and dense RGB for depth completion **145**
A.1 Architecture details 145

B Multi-view PointNet for 3D scene understanding **147**
B.1 2D Encoder Decoder Architecture 147
B.2 Additional Ablation Studies 147

C 2D-3D Cross-Modal Unsupervised Domain Adaptation **149**
C.1 Dataset Splits 149
C.1.1 nuScenes 149
C.1.2 A2D2 and SemanticKITTI 150

Introduction

French Summary of the Chapter "Introduction"

D'importantes avancées dans le domaine de recherche de la conduite autonome ont permis la réalisation de démonstrateurs impressionants. Ces avancées ont été réalisées d'une part grâce à l'amélioration des capteurs et d'autre part grâce à de nouveaux algorithmes, en particulier l'apprentissage profond. Néanmoins, il reste encore de nombreux challenges à résoudre. Dans cette thèse, nous abordons les défis de la rareté des annotations et la fusion de données hétérogènes telles que les nuages de points 3D et images 2D.

Chapter 1

Introduction

Contents

1.1 Goals 5
1.2 Autonomous driving context 7
1.3 Thesis Structure 9

Already in 1994, researchers of the Prometheus project showcased automated driving on motorways. During three DARPA Grand Challenges from 2004 to 2007, different teams competed with their fully-driverless cars to navigate in desert and controlled urban settings. This was seen as proof that self-driving cars are feasible and sparked great optimism that the technology would quickly become available. Now, a decade since the start of the Google driverless car project in 2009, first fleets of robot taxis give rides to customers in geographically restricted areas[1]. In a parallel development, privately owned vehicles are equipped with an increasing number of Advanced Driver-Assistance Systems (ADAS) such as active steering, speed control and lane change on highways, as well as automated parking. While these features work increasingly well, the driver still has to monitor the car at all times. The first system where the driver can hand over responsibility to the car in the limited (but often encountered) scenario of highway traffic jams is ready, but still needs to be certified[2].

In light of this progress, large-scale rollout of automated driving promises to improve road safety and decrease transportation cost. However, experts remain uncertain how much time it will take until driverless cars can commercially operate in urban scenarios[3].

To better understand the technological difficulties that we still face, let us briefly introduce the pipeline of autonomous driving, traditionally split into the modules of perception, planning and control. Cameras, LiDARs, radars and ultrasonic sensors scan the surroundings of the car and perception algorithms analyze the raw sensor data to detect other cars, pedestrians, driving lanes etc. Using the output from the perception module, the planning module forecasts the intentions of other road users, i.e. future trajectories, on the basis of which the appropriate ego trajectory is chosen. In the control module, the ego trajectory is then translated to the actuators.

I think that today's challenges of autonomous driving mainly lie in perception and planning. Perception needs to achieve scene understanding by fusing sensor data of different nature, while being robust to adverse conditions such as night driving, rain and snow. Moreover, there is great variability in possible traffic situations with a long tail of rare events. It can prove prohibitively expensive to annotate this data, as it needs to be done by hand. For the planning module, unpredictability of human behavior is a big challenge, especially if traffic rules are disrespected. However, forecasting the actions of other road users is crucial to take reasonable decisions.

In the recent years, deep learning has brought tremendous progress to machine learning and especially computer vision. At the forefront was the

[1] https://venturebeat.com/2019/12/05/waymo-one-ios-app-launch/

[2] https://www.dw.com/en/autonomous-cars-when-will-our-cars-finally-really-drive-us/a-49620020

[3] https://www.nytimes.com/2019/07/17/business/self-driving-autonomous-cars.html

development of methods for image classification (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; He et al., 2016), semantic segmentation (Long et al., 2015a) and object detection (Ren et al., 2015). Since then, we have witnessed the adaptation of these image-focused techniques to 3D point cloud data (Qi et al., 2017a; Graham and van der Maaten, 2017). In another line of work, the dependence on labels have successfully been reduced, for example through unsupervised representation learning (Doersch et al., 2015) or the knowledge transfer from a labeled to an unlabeled and visually different data domain (Ganin et al., 2016). Moreover, the application of deep learning to reinforcement learning has enabled machines to play Atari games (Mnih et al., 2013) and beat human players at Go (Silver et al., 2016) and Starcraft II (Vinyals et al., 2019).

These recent scientific advances have already helped to enable today's successes of autonomous driving to a notable extent, but further research is key to address the remaining challenges. Moreover, interesting research problems are triggered by autonomous driving and can potentially be transferred to other applications such as robotics for industry and farming, as well as mixed reality that uses headsets to superpose a virtual layer on the real world.

In the remainder of this chapter, we present our research goals in Section 1.1, introduce the autonomous driving context in Section 1.2 and then outline the organization of this document in Section 1.3.

1.1 Goals

In this thesis, we address three main challenges. First, we aim to simplify the autonomous driving pipeline and reduce the necessity for abstract labels using an end-to-end driving approach. Second, we focus on perception to fuse data from multiple sensors for more accurate and robust scene understanding. Third, we aim to reduce the reliance on labels when training perception systems by leveraging multi-modality.

End-to-end driving. While the decomposition of the autonomous driving pipeline into perception, planning and control modules allows to optimize each sub-problem independently, it requires to make hand-designed abstractions at intermediate outputs, e.g. the perception module produces abstract bounding boxes around detected objects. However, if the abstract intermediate outputs from upstream modules are not expressive enough to represent the environment, the downstream modules are unable to work properly. We aim to address this problem with the approach of end-to-end driving where a neural network maps sensor inputs directly to control outputs, i.e. integrating all modules into a single one. This simplifies the pipeline and eliminates the need for costly labels of intermediate outputs such as bounding boxes

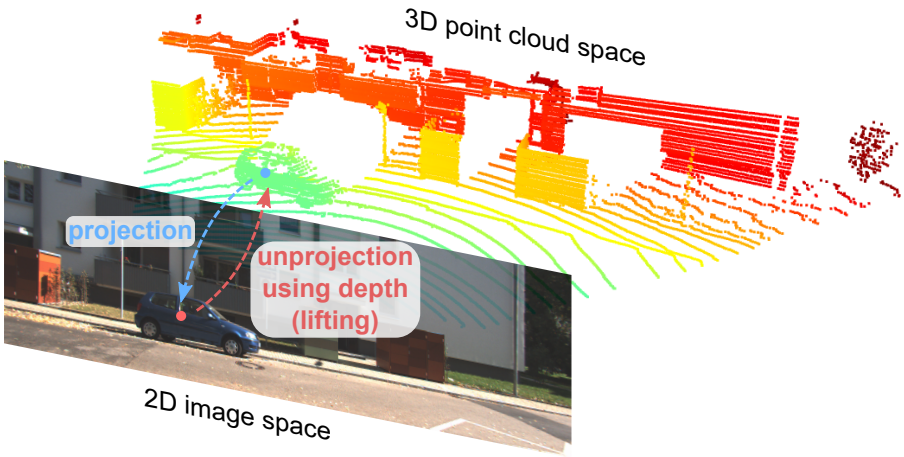


Figure 1.1: Projection of 3D points into the image and unprojection of 2D image pixels to 3D using the pixel depth information. Data from Kitti dataset (Geiger et al., 2013).

of detected objects, because we can directly train the network end-to-end, judging only the final control output. Analogously in computer vision, great performance improvements have been achieved by shifting from modular approaches, i.e. computing intermediate features (e.g. Histogram of Oriented Gradients) followed by a learning based classifier, to Convolutional Neural Networks (CNNs) that are trained end-to-end. While the task is immensely more complex, there is hope that training a neural network for the driving task in an end-to-end fashion can also improve performance[4].

2D-3D fusion. As discussed earlier, self-driving cars are equipped with many sensors that analyze the surrounding environment. LiDAR and camera are the sensors with highest resolution and are complementary: while LiDAR outputs a 3D point cloud, measuring scene *geometry*, camera outputs 2D images, capturing *visual* appearance. For example, camera is crucial to analyze street signs and other objects that can not be discriminated based on geometry only. However, the active laser scanning technology of LiDAR also works well at night, while passive camera is highly impacted by external lighting conditions. Our goal is to fuse those complementary sensors to enable more robust and better performing perception systems. In spite of recent progress in 2D and 3D deep learning, many questions remain regarding the combination of multi-modal data, especially how to learn joint features given their heterogeneous representation spaces.

Thanks to sensor calibration we can compute a correspondence mapping

[4] https://slideslive.com/38917941/imitation-prediction-and-modelbased-reinforcement-learning-for-autonomous-driving

between a point in 2D and 3D space as shown in Figure 1.1. When projecting the point cloud into image space, we lose one dimension and thus information about the 3D structure about the scene. To carry out unprojection or 2D-3D lifting from 2D pixel to 3D point, we need the corresponding depth information for that pixel. This is a challenge, because usually we do not have the depth information for all pixels due to the relative *sparsity* of the depth sensor once projected into image space. In summary, finding a space in which to fuse 2D-3D data is not straightforward and depends on the data and task.

Unsupervised multi-modal learning. The supervised training of 2D-3D fusion networks requires a lot of labeled data. However, annotation by hand is tedious and costly. While semantic labeling of an image in the Cityscapes dataset "required more than 1.5h on average" (Cordts et al., 2016), the semantic labeling of point clouds in the SemanticKITTI dataset (Behley et al., 2019) of a 100x100m tile (seen from birdview perspective) took a staggering 4.5h. Unfortunately, networks that are trained on a labeled dataset still fail to generalize to different environments due to *domain shift* between data distributions. With multi-modal 2D-3D data, especially when performing fusion, the problem becomes more complex as the domain shift can be different in each modality, i.e. sensing technology such as camera and the LiDAR might be affected differently, for instance when shifting from day to night, given that LiDAR has better night-vision.

We argue that the co-registration of multiple sensors provides valuable information when labels are scarce. Indeed, audio-visual correspondence learning in videos has shown encouraging results (Arandjelovic and Zisserman, 2017). We aim at exploiting RGB+LiDAR multi-modality as unsupervised learning signal.

1.2 Autonomous driving context

This three-year thesis was conducted with Inria RITS team and the company Valeo. Both have a long-standing history of collaborations through joint projects and Valeo's financing of PhD students.

As a robotics team specialized in autonomous vehicles, the key research areas of Inria RITS (Robotics for Intelligent Transportation Systems) lie in perception, decision making, control and large-scale traffic modeling and optimization. On the perception part, most of the research focuses on scene understanding from vision sensors (cameras, LiDARs, etc.).

Among other products, the automotive supplier Valeo produces sensors such as cameras, LiDARs, radars and ultrasonic sensors and develops associated software systems. This thesis was initiated by the Driving Assistance Research (DAR) team that focuses on research and development in

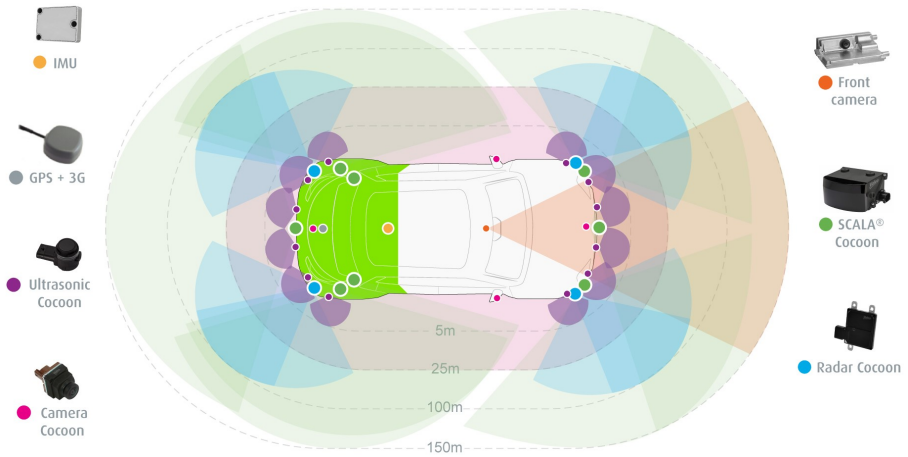


Figure 1.2: Sensor setup of the Valeo Drive4U prototype. Source: Valeo.

autonomous driving and Advanced Driver-Assistance Systems (ADAS). I also joined Valeo.ai after its creation in 2018. This industrial open research lab investigates and publishes mainly in the field of multi-sensor perception, domain adaptation and uncertainty estimation.

Sensors. To understand their surrounding environment, self-driving cars rely on a suite of complementary sensors. An example setup of a Valeo self-driving prototype car with respective fields of view is depicted in Figure 1.2. While the setup may vary, most self-driving car prototypes are equipped with similar sensors. Camera is the closest sensor to our human eyes, essential for capturing *visual* cues. Note the high-resolution front camera with restricted field of view for long-range vision at high speeds and 'cocoon' cameras with wide viewing angle to cover the complete surroundings close to the car. In contrast to the passive camera, LiDAR uses laser beams for active scanning with time-of-flight measurement to infer distances to objects, capturing the *geometry*. Note that, even though not in the Valeo setup shown in Figure 1.2, 360° LiDAR is often used, producing a complete scan of the car's surroundings. Instead of high-frequency light waves used by LiDAR, automotive radars typically emit waves at around 77 GHz and leverage the Doppler effect which permits accurate velocity measurement of moving objects, but at low resolution in terms of location. Ultrasonic sensors are much more limited than the other sensor types, because the comparably low frequencies of 40-70kHz right above the human hearing range of 20Hz to 20kHz only enable small fields of view and are only precise at low speeds. Therefore, many of them are installed around the vehicle and they are mostly used for parking applications. In light of these limitations, we focus on LiDAR and camera in this thesis, as they are the most informative sensors due to their high resolutions. Note the overlapping fields of view in Figure 1.2 which

favors multi-sensor fusion.

Apart from perception sensors, self-driving cars are also equipped with a Global Positioning System (GPS) and an Inertial Measurement Unit (IMU) to localize in a high definition map, as well as communication modules to allow data exchange with infrastructure and other cars.

Towards integrated data-driven pipelines. With the advent of deep learning and the intention to tackle complex urban driving, there has been a paradigm shift in soft- and hardware architectures of perception systems in autonomous driving.

Traditional perception architectures rely on lightweight recognition algorithms that can be optimized to fit into economical processing units, directly integrated into each sensor. Consequently, all sensors already produce *high-level* output, e.g. bounding boxes for object detection, which can then be fused centrally, e.g. with a Kalman filter.

Today the trend goes towards raw data fusion, where the sensors simply provide the captured data, i.e. point cloud and image, to a computationally powerful central processing unit, usually carrying a GPU to accelerate neural networks. A multi-modal perception algorithm running on the central processing unit can then exploit the complementarity of the sensors, e.g. by carrying out *feature fusion* inside a neural network.

In this thesis, we are interested in new paradigm, i.e. in designing data-driven fusion methods using neural networks while exploiting multi-modality through feature fusion.

1.3 Thesis Structure

The rest of this document is organized as follows.

In Chapter 2, aside from the main topic of this thesis, we address the challenge of end-to-end driving, i.e. learning to drive using a single neural network that maps sensor input (the camera image) to control output (throttle, break, steering). A commonly encountered problem in the imitation learning technique is distribution mismatch between the expert data seen during training and neural network driving at test time. To tackle this issue, we leverage *Reinforcement Learning (RL)* where an agent directly learns to drive by itself without the need for an expert. As RL relies on trial-and-error learning, it is possibly dangerous in the real world, as it could lead to crashing a car. Therefore we resort to training in a simulator, but we choose a realistic one in an attempt to minimize domain gap between training and the real world.

In Chapter 3 we introduce our first line of work regarding 2D-3D fusion. To represent point cloud and image in the same space, we *project the 3D point cloud into 2D image space* resulting in a sparse depth map. We focus

on the task of *depth completion*, i.e. filling the holes in the sparse depth map and thereby enhancing LiDAR resolution to image level. To achieve that, we investigate how 2D CNNs cope with sparse input data and develop an architecture that fuses dense RGB and sparse depth, effectively using RGB guidance for depth up-sampling, especially helpful at low LiDAR resolution. Moreover, we argue that this data-driven approach can also leverage knowledge about general scene structure. However, there are limitations to our method due to 3D-2D projection, because the loss of a spatial dimension leads to decreased precision.

In our second line of work, presented in Chapter 4, we use *multi-view* images and a global point cloud as input data for the task of 3D semantic segmentation of indoor scenes. We assume that it is preferable to solve a 3D task using a 3D data representation. Therefore, we *lift 2D information to 3D*, rather than using 3D to 2D projection like in Chapter 3 which leads to a loss in precision. To still be able to process images in their natural 2D space with efficient CNNs, we perform 2D-3D lifting of features after the 2D CNN is applied, instead of lifting the original RGB values. As we consider multiple views, we can represent them all in the same 3D space, which enables efficient fusion and subsequent semantic segmentation with a 3D point network. We show in experiments that our 2D-3D fusion strategy outperforms the 3D baseline where only *colors* from images, but not *features* like in our approach, are lifted to the point cloud. Like in Chapter 3, we find that 2D-3D fusion increases robustness to low point cloud resolution.

In Chapter 5, we introduce the novel task of *cross-modal unsupervised domain adaptation,* where the aim is to address domain shift between a labeled source and an unlabeled target dataset that both hold multi-modal data. We propose *mutual mimicking* between image and point cloud data to enable 2D-3D cross-modal learning and effectively reduce the discrepancy between source and target distributions. We evaluate our method on three different UDA scenarios using recent driving datasets and find cross-modal learning to exhibit domain adaptation capabilities comparable to the uni-modal state-of-the-art techniques. Additionally, as our technique is complementary to existing approaches, we can achieve best performance by combining cross-modal learning with other techniques such as pseudo-labeling.

Finally, Chapter 6 summarizes our contributions and gives an outlook on future work.

Conduite de bout en bout avec apprentissage par renforcement profond

French Summary of the Chapter "End-to-End Driving with Deep Reinforcement Learning"

Nous adoptons une stratégie de *conduite de bout en bout* où un réseau de neurones est entraîné pour directement traduire l'entrée capteur (image caméra) en contrôles-commandes, ce qui rend cette approche indépendante des annotations dans le domaine visuel. Pour ce faire, nous utilisons l'apprentissage par renforcement profond où l'algorithme apprend de la récompense, obtenue par interaction avec un simulateur réaliste. Nous proposons de nouvelles stratégies d'entraînement et fonctions de récompense pour une meilleure conduite et une convergence plus rapide.

Chapter 2

End-to-End Driving with Deep Reinforcement Learning

The contributions of this chapter were published in a short workshop paper (Perot et al., 2017) and in a conference paper (Jaritz et al., 2018a):

Perot, E., Jaritz, M., Toromanoff, M., and de Charette, R. (2017). End-to-end driving in a realistic racing game with deep reinforcement learning. In *CVPR Workshop 2017*.
1-minute driving video: https://youtu.be/e9jk-lBWFlw

Jaritz, M., de Charette, R., Toromanoff, M., Perot, E., and Nashashibi, F. (2018a). End-to-end race driving with deep reinforcement learning. In *ICRA 2018*.
3-minute explanation video: https://youtu.be/AF0sryuSHdY

This project was initiated when I worked at Valeo before starting this thesis, with the goal to demonstrate Deep Reinforcement Learning for Autonomous Driving in a realistic simulator at the Consumer Electronics Show 2017. During the first months of my thesis with Inria and Valeo, the project was extended with more emphasis on the scientific contributions.

Note that this chapter is quite different from the others, because it tackles the complete autonomous driving task, while the remaining chapters focus on perception.

Contents

2.1 Introduction **15**
2.2 Reinforcement learning background **18**
2.3 Related Work **20**
2.4 Method **23**
2.4.1 Reinforcement learning framework 24
2.4.2 Learning strategy 27
2.5 Experiments **31**
2.5.1 Training setup 31
2.5.2 Metrics 31
2.5.3 Performance evaluation 32
2.5.4 Ablation studies 34
2.5.5 Generalization 37
2.6 Discussion **39**
2.7 Conclusion **40**

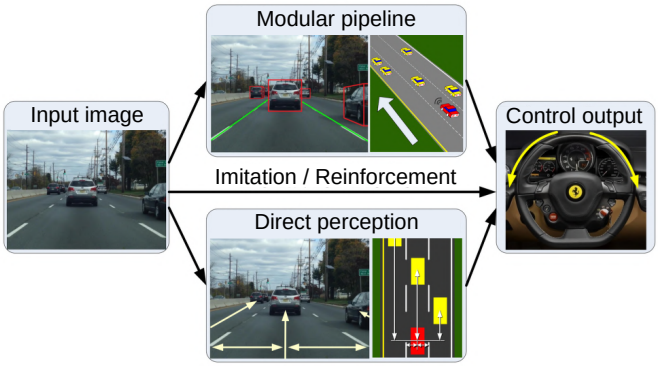


Figure 2.1: Different approaches for vision-based driving. The commonly used *modular pipeline* approach splits driving into separate algorithms for perception, planning and control. By contrast, the *end-to-end* approach directly maps sensor input to control output. *Direct perception* is a hybrid approach between the two preceding ones. Source: Chen et al. (2015), modified.

2.1 Introduction

The well-established approach of a *modular pipeline* (Sun et al., 2006; Urmson et al., 2008; Montemerlo et al., 2008) divides the autonomous driving task into three parts – perception, planning, control. For example, the perception module estimates the driving lanes and bounding boxes of other cars, as depicted in Figure 2.1 in the top middle, and provides this information to the planning module. The planning module forecasts the intention of other road users, computes a trajectory and passes it to the control module which in turn calculates the final control output. Today, most autonomous cars use this paradigm as it enables the decomposition of a problem into simpler sub-problems and allows for interpretation of intermediate outputs along the pipeline.

However, there are limitations to this approach, especially when considering the variety of complex urban driving scenarios, because the abstract intermediate outputs along the modular pipeline turn out to be information bottlenecks. For example, the perception module provides the planning module with bounding boxes of detected cars and pedestrians, but other information that might help forecasting the trajectories of other road users is lost. Consider a pedestrian waving at a bus driver from the other side of the street signaling him to wait. This informs us that the pedestrian will probably cross the street even if there is no crosswalk and our car should take this into account. However, the planner is unable to do so, because an abstract bounding box from the perception module does not convey enough information.

End-to-end driving tries to overcome the difficulties of the modular pipeline approach by training a neural network that maps raw sensor input (e.g. image) directly to output controls (e.g. throttle, break, steering angle), as shown in Figure 2.1. Information bottlenecks are reduced, because there is only a 'single module'. Thus, complex situations with detailed information can be encoded in high-dimensional feature space and preserved while being passed on from layer to layer inside the neural network. Moreover, there are no hand-written rules, because the network is trained end-to-end. In the following, we introduce the existing end-to-end approaches, grouped into three categories: imitation learning, reinforcement learning and direct perception. Note that direct perception does not directly predict control output, but can still be seen as end-to-end technique.

Imitation learning tries to clone the human driver by leveraging expert driving data in a supervised setting (Pomerleau, 1989; Bojarski et al., 2016). No annotations are necessary, because the recorded driver's actions, i.e. how the driver applies throttle, brake and steering at each time step, serve directly as labels for the corresponding raw sensor data (e.g. the camera image). This allows for the collection of large amounts of training data at low cost. However, there is the problem of *distribution mismatch*: the expert will rarely, if ever, encounter situations of failure (e.g. significant deviation from lane, stopping too close behind the vehicle in the front at the red light, etc.) and thus, the self-driving system can not learn to react accordingly in such a failure situation as this is missing from the training set. Moreover, the performance of imitation learning is always limited by the quality of demonstrations. For instance, it is not possible to train a smooth driving system with training data from a roughly driving expert. Unlike for other supervised tasks, there isn't a unique ground truth for driving, i.e. different driving styles exist and are all valid. In short, imitation learns to mimic the driving style in the training set.

Reinforcement learning (RL) (Sutton and Barto, 1998), or Deep RL (DRL) when using a deep neural network, differs significantly from imitation learning in that it is trained in a self-supervised fashion. It consists of an agent that learns by itself via interaction with the environment. The trial-and-error learning behavior of RL is dangerous in the real world where accidents are to be avoided at all cost. For this reason, RL is most often carried out in simulators. Learning is based on reward instead of labels, where the goal is to maximize reward, accumulated over time. The reward can be sparsely distributed, for example when achieving a goal such as successfully completing a turn right maneuver. Reward can also be negative, i.e. a penalty, for instance when the car departs from the lane. Hence, reward is different from labels and not tied to the local action, but rather to the overall achievement. Unlike imitation learning, RL can prevent distribution mismatch between the situations encountered during training and test, i.e. the agent can choose bad actions during training and learn how to counterbal-

(a) TORCS (b) WRC6

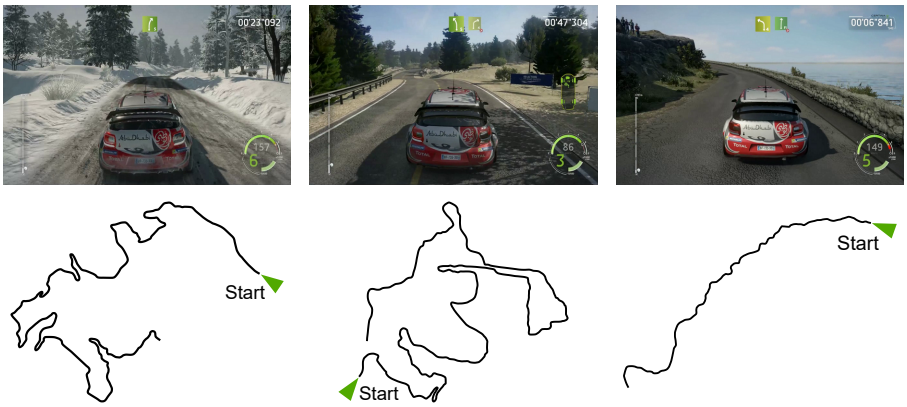
Figure 2.2: Screenshots of race driving simulators. (a) TORCS has basic physics and graphics, and was used in related reinforcement learning works (Mnih et al., 2016; Lillicrap et al., 2016). (b) WRC6 is a modern rally racing game, released in 2016, that tries to make the physics and graphics as realistic as possible. We choose this simulator to facilitate generalization to real data.

ance them. However, there are also drawbacks to RL. Training time is very long, because reward is a weaker and sparser learning signal than explicit labels in supervised learning. Moreover, transferring from simulated to real environments is a challenge, especially if the simulators lack realism such as the commonly used TORCS environment (Wymann et al., 2000), depicted in Figure 2.2a. In this chapter, we use RL to tackle end-to-end driving and introduce the general background of RL more extensively in Section 2.2.

Direct perception is a hybrid approach between end-to-end driving and modular pipelines (Chen et al., 2015). The idea is to learn an intermediate interpretable representation that is sufficient to control a vehicle. For example, a network could directly predict the distance to the vehicle ahead and feed it to a controller that uses this distance to compute the appropriate low-level control commands. However, even if direct perception can potentially improve interpretability, it still requires a rule-based controller.

In this chapter, we try to solve the race driving task by using the front camera image as input and directly predicting low-level controls (throttle, brake, hand brake, steering). To that end, we employ the state-of-the-art deep reinforcement learning algorithm A3C (Mnih et al., 2016) and modify the action space, reward function, network architecture and agent initialization procedure to optimize for race driving and to speed up convergence. We use the video game World Rally Championship 6 (WRC6)[1] as simulation environment. A side-by-side comparison of TORCS and WRC6 is shown in Figure 2.2. WRC6 is much more realistic than TORCS, as WRC6 was

[1] http://www.wrcthegame.com



(a) Snow (SE) 11.61km (b) Mountain (CN) 13.34km (c) Coast (UK) 4.59km

Figure 2.3: Screenshots and outlines of the three training tracks in WRC6. There are important differences in visual aspects and road layout.

designed to be as genuine as possible in terms of both physics (mass transfer, road adherence of the tires, etc.) and graphics. By using such a highly realistic simulator, we hope to facilitate the transfer from simulation to real world driving.

We train on three tracks with a total length of 29.6km, shown in Figure 2.3. Visual appearances (snow, mountain, coast) and physics (road adherence) vary greatly across these tracks. Using distributed learning, we train on the three tracks in parallel, learning a single policy. Our contributions can be summarized as:

- We apply DRL to the realistic race driving simulator WRC6.
- We propose new training strategies to maximize benefits of asynchronous learning (multiple tracks, agent initialization).
- We design new reward functions that demonstrate faster convergence and better driving.

Learning a single policy, we successfully drive on all training tracks in WRC6. Moreover, we observe some degree of generalization when testing the resulting policy on an unseen track in WRC6 or on real videos. For the latter, we conduct open loop testing and qualitatively observe the prediction of reasonable actions to follow the road, avoid oncoming traffic etc.

2.2 Reinforcement learning background

In the common RL setup, depicted in Figure 2.4, an agent interacts with an environment at discrete time steps t by receiving the state s_t, on which basis

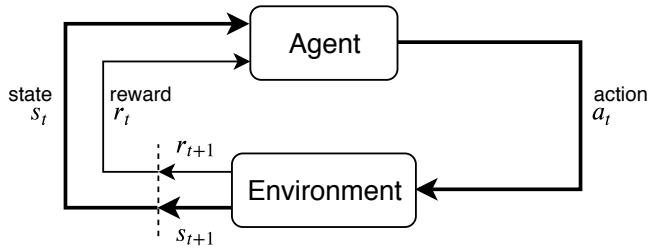


Figure 2.4: Reinforcement learning setup where an agent learns by interaction with an environment. The agent computes an action a_t based on the current state s_t. The environment executes action a_t and moves to a new state s_{t+1} and yield a reward r_{t+1}. The agent optimizes its policy, i.e. how to choose actions, by seeking to maximize reward. Adapted from: Sutton and Barto (1998).

it selects an action a_t as a function of policy π with probability $\pi(a_t|s_t)$ and sends it to the environment where a_t is executed and the next state s_{t+1} is reached with associated reward r_{t+1}. Both, state s_{t+1} and reward r_{t+1}, are returned to the agent which allows the process to start over. The discounted return corresponds to the sum of rewards that is gained taking infinite time steps and is to be maximized by the agent. It writes:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{2.1}$$

where $\gamma \in [0, 1[$ is the discount factor. Hypothetically, we could set $\gamma = 1$, so that there would be no decay and future rewards are as important as the current reward. However, this creates a mathematical problem, because the sum could go to infinity for positive rewards. Setting $\gamma < 1$ makes the sum of rewards finite and helps convergence of reinforcement algorithms. A typical value is $\gamma = 0.99$ which decays the rewards over time, making an agent slightly short-sighted in desiring to collecting rewards earlier, rather than later.

RL algorithms can be divided into three different families. *Policy-based* RL methods directly try to estimate the policy $\pi(a|s)$. Thus, for each state s, the policy $\pi(a|s)$ tells us which action we should take. In practice, policy-based RL methods are often coupled with an actor-critic approach to stabilize training, which we will introduce in detail in Section 2.4.1, along with the actor-critic A3C (Mnih et al., 2016), used in this work.

Value-based RL algorithms or Q-learning, seek to approximate the optimal action-value function

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\pi}[R_t|s_0 = s, a_0 = a] \tag{2.2}$$

which gives the expected return for taking action a in current state s and acting according to the optimal policy thereafter. Value-based methods do not estimate the policy $\pi(a|s)$. Instead, they choose the optimal action $a^*(s)$ that maximizes the action-value function

$$a^*(s) = \arg\max_a Q^*(s, a). \tag{2.3}$$

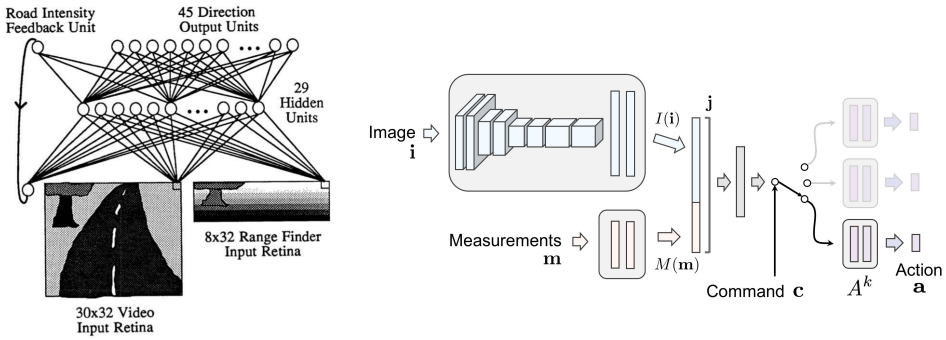
The third family are *model-based* RL methods which are mostly used if one has access to a model of the environment. In this case, one can use the model to plan several steps ahead before choosing the action. However, while the model is completely known for simple environments such as boardgames (Go, chess, etc.), modeling driving is very complex.

The actions in RL can be continuous or discrete. In the discrete case, action prediction amounts to a classification problem. The set of available actions is referred to as *action space*. When training an RL algorithm, an *episode* refers to a whole sequence from initial to terminal state (e.g. from game start to game over).

2.3 Related Work

In the following we present related works in imitation learning and Deep Reinforcement Learning (DRL). While we deliberately limit ourselves to works applied to end-to-end driving in imitation learning, we take a slightly larger scope in RL: we first detail the methods of two representative DRL works and then introduce those applied to end-to-end driving.

Imitation learning. The seminal work of Pomerleau (1989), depicted in Figure 2.5a, already achieved astonishing results at the time: from camera and LiDAR inputs, a neural network with two fully-connected layers predicted the road direction to follow. More recently, but in the same spirit, Bojarski et al. (2016) train an eight layer CNN to learn the lateral control from a front view camera image, using the steering angle from a real driver as ground truth. They use 72h of training data, with homographic interpolation of three forward-facing cameras to generate novel viewpoints. This helps to address distribution mismatch by simulating failure situations (deviations from the road). Pomerleau (1989) and Bojarski et al. (2016) achieve following a single road. However, to navigate intersections, Codevilla et al. (2018) introduce conditional imitation learning where the network takes high-level driving commands, such as 'turn left', 'turn right' or 'go straight' as additional input. The architecture is shown in Figure 2.5b: a CNN takes the front camera image as input and produces features $I(\mathbf{i})$ that are concatenated with features $M(\mathbf{m})$ which are computed from measurements (current speed of the car) with fully-connected layers. In the



(a) Pomerleau (1989) (b) Codevilla et al. (2018)

Figure 2.5: End-to-end driving with imitation learning. (a) Pomerleau (1989) use a neural network with only two fully-connected layers takes image and LiDAR point cloud as inputs and directly predicts the road direction to follow. (b) Codevilla et al. (2018) concatenate CNN image features $I(\mathbf{i})$ and measurement features $M(\mathbf{m})$ to feature vector $\mathbf{j}$. The input command $\mathbf{c}$ switches to the corresponding fully-connected layers of the action prediction branches A^k, specialized in different commands, for example 'turn left', 'turn right' or 'go straight'.

following, $I(\mathbf{i})$ and $M(\mathbf{m})$ are concatenated to the feature vector $\mathbf{j}$ which is shared between different branches A^k that are composed of fully-connected layers and specialize in the different possible commands $\mathbf{c}$, e.g. 'turn left'. The command $\mathbf{c}$ acts as switch to choose the correct branch A^k and can also be given at test time to influence the driving.

Chen et al. (2019a) use privileged information, the complete environment state in form of a map, to train a teacher which, in a second step, supervises a student that has only access to the image.

Deep reinforcement learning. In the following, we introduce the representative works of DQN (Mnih et al., 2013) and DDPG (Lillicrap et al., 2016), list works with applications to video games and robotics, and finally detail DRL works applied to autonomous driving.

DQN (Mnih et al., 2013) is the first combination of modern deep neural networks with reinforcement learning which has given rise to the term of *deep* reinforcement learning. DQN uses Q-learning (Watkins and Dayan, 1992) which approximates the optimal action-value function $Q^*(s, a)$. In DQN, the approximation is carried out with a neural network with parameters θ_i:

$$Q(s, a; \theta_i) \approx Q^*(s, a) \tag{2.4}$$

where i refers to each iteration of the training algorithm, i.e. the weights θ_i are updated at each iteration.

Mnih et al. (2013) use a replay buffer $\mathcal{B}$ where they store experiences $e = (s, a, r, s')$ where state s' results from taking action a in state s. As opposed to the *policy-based* family that requires *on-policy* updates, DQN, which is part of the *value-based* family, is able to learn *off-policy* from arbitrary experiences. Thus, during each training iteration i in DQN, a mini-batch of experiences can randomly be sampled from the replay buffer: $(s, a, r, s') \sim U(\mathcal{B})$. This way, experiences are decorrelated, as opposed to online learning without buffer where experiences come in order and are correlated. Moreover, the replay buffer allows for greater data-efficiency, because experiences can be used multiple times to update the weights of the network before being discarded.

The regression of the action-value function $Q(s, a; \theta_i)$ is achieved with an $\mathcal{L}_2$ loss between $Q(s, a; \theta_i)$ and the target y_i:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{B})}\Big[\Big(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{y_i} - Q(s, a; \theta_i)\Big)^2\Big] \tag{2.5}$$

where the target y_i is computed as sum of reward r and the value of the Q-function to approximate the return of the remaining time steps. Note that the Q-function is used in the regression and the target y_i. This can pose a stability problem during training, because there can be a feedback loop where the Q-network wrongly reinforces itself in the belief that some state-action pair is very valuable. To mediate this, Mnih et al. (2013) use a *target* network with θ_i^- of an *earlier* iteration to compute the target y_i. Although this slows down training, it effectively reduces the risk of feedback and stabilizes training.

In the loss function of Equation 2.5 and also to choose actions at inference time, a maximum of the Q-function has to be taken over all possible actions. This is straightforward when the action space is discrete and thus finite. However, it becomes intractable when the action space is continuous. DDPG (Lillicrap et al., 2016) addresses this problem by learning a target policy $\mu(s; \theta_{\text{targ}})$, parameterized by θ_{targ}, that estimates the continuous action which maximizes Q. Thus, the Q-function only needs to be evaluated once using the predicted action $\mu(s'; \theta_{\text{targ}})$. Hence, the DDPG variant of Equation 2.5 writes:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{B})}\Big[\Big(r + \gamma Q(s', \mu(s'; \theta_{\text{targ}}); \theta_i^-) - Q(s, a; \theta_i)\Big)^2\Big]. \tag{2.6}$$

Note that, there is no max anymore.

Many DRL algorithms are evaluated on Atari games, including DQN (Mnih et al., 2013), A3C (Mnih et al., 2016), IQN (Dabney et al., 2018) or on the robotic simulator MuJoCo (Todorov et al., 2012), including DDPG

(Lillicrap et al., 2016), TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017) and SAC (Haarnoja et al., 2018).

Another line of work applies DRL to real robotic arms (Levine et al., 2016) and has recently been extended to parallel training on an ensemble of robot arms (Gu et al., 2017; Kalashnikov et al., 2018; Levine et al., 2018).

Opposed to that, only few DRL works address the end-to-end driving task. DDPG (Lillicrap et al., 2016) used a continuous and A3C (Mnih et al., 2016) a discrete action space to learn to drive in the TORCS race driving simulator (Wymann et al., 2000). The reward in both works is computed as the car's velocity projected onto the direction of the road. A practical problem of training RL algorithms is that training examples are correlated when they originate from the same agent evolving in an environment. To address this, DDPG (Lillicrap et al., 2016) uses a replay buffer from which diverse past examples can be recalled, whereas A3C leverages asynchronous learning with multiple agents that evolve in different instances of the environment in parallel. In this work, we use A3C (Mnih et al., 2016), because it was the state-of-the-art at the time of our contribution and training is more stable than in DDPG. Moreover, the asynchronous learning framework of A3C facilitates distribution across multiple machines for faster training.

Kendall et al. (2019) were the first to apply RL to a real car. They use DDPG (Lillicrap et al., 2016) for the rather simple task of lateral control to follow a country road. The reward was computed as the distance that was covered without human intervention.

The preceding RL works in end-to-end driving (Lillicrap et al., 2016; Mnih et al., 2016; Kendall et al., 2019) use extremely shallow CNNs of 3 or 4 layers to encode the information from the image, because they are sufficient for simple road following and because a larger network would take much longer to train. In order to benefit from deeper networks, commonly used in perception systems, Toromanoff et al. (2020) split training into two stages. First, they supervisedly pretrain the perception part of the framework, a CNN with ResNet18 backbone (He et al., 2016), on proxy tasks such as semantic segmentation and traffic light state prediction. Second, they freeze the weights of the visual encoder and carry out RL of the vehicle control part of the framework which consists of fully-connected layers.

2.4 Method

In this chapter, we aim at learning end-to-end rally driving, where visual and physical conditions change on each track, e.g. slippery dirt road in snowy winter forest or dry tarmac in sunny weather. This task is challenging because we seek to not only learn how to drive but to implicitly learn the dynamics of the car depending on the conditions (varying physics in each track).

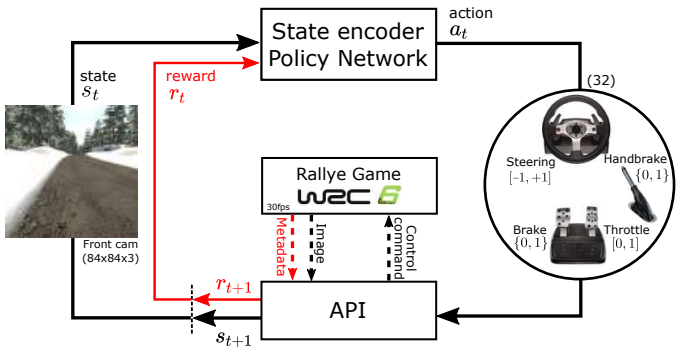


Figure 2.6: Overview of our end-to-end driving framework using the WRC6 rally environment (red = for training only). The state encoder tries to learn the optimal control commands, using only 84x84 front view images and speed. The stochastic game environment is complex with realistic physics and graphics.

We learn full control – steering, brake, throttle and even hand brake to enforce drifting – to drive in the realistic rally racing game WRC6 with an API to communicate images, metadata and control decisions between RL algorithm and simulator. The WRC6 communication API was specifically built for our need in the context of a contractual collaboration with Kylotonn, the game developing company. The overall pipeline is depicted in Figure 2.6. At every time-step, the algorithm receives the state of the game s_t, acts on the car through an action a_t (a combination of control commands which is executed in the simulator) and finally gets back a reward r_{t+1}. We employ the RL algorithm A3C (Mnih et al., 2016) to optimize a driving policy that outputs the action probabilities for vehicle control, using only the RGB front view image as input.

In the following Section 2.4.1, we introduce A3C (Mnih et al., 2016) and explain how we carried out distributed learning on multiple machines. In Section 2.4.2, we detail our learning strategy that consists of specific design choices for the rally driving task.

2.4.1 Reinforcement learning framework

Asynchronous Advantage Actor Critic (A3C). In this work, we employ A3C (Mnih et al., 2016) which is a member of the policy-based RL family and uses an actor-critic approach. The actor chooses the action a in a certain state s and is formalized by policy $\pi(a|s)$. While the policy outputs action probabilities, the critic estimates how good a certain state s is with the value function $V^\pi(s)$. It computes the expected return that is obtained departing from state s and always acting according to policy π:

$$V^\pi(s) = \mathbb{E}_\pi[R_t|s_0 = s]. \tag{2.7}$$

The policy $\pi_\theta(a_t|s_t;\theta)$ is parameterized by θ, i.e. the weights of a neural network. To optimize π_θ, gradient ascent is performed to maximize the expectation of the discounted reward $\mathbb{E}[R_t]$, introduced in Equation 2.1. This means that we update policy π_θ, such that, in a state s, actions that bring high reward are made more likely and actions that bring low reward less likely. The REINFORCE method (Williams, 1992) estimates the policy gradient $\nabla_\theta \mathbb{E}[R_t]$ as

$$\nabla_\theta \log \pi_\theta(a_t|s_t;\theta) R_t. \tag{2.8}$$

To reduce the variance of this estimate, one can subtract a baseline $b_t(s_t)$, i.e. a known policy, from R_t. This can be written as

$$\nabla_\theta \log \pi_\theta(a_t|s_t;\theta)(R_t - b_t(s_t)). \tag{2.9}$$

The difference $R_t - b_t(s_t)$ measures if our policy performs better or worse than the baseline $b_t(s_t)$ and scales the policy gradient $\nabla_\theta \log \pi_\theta(a_t|s_t;\theta)$. In actor-critics, the difference $R_t - b_t(s_t)$ is replaced by the advantage function

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t) \tag{2.10}$$

where the Q-function $Q^\pi(a_t, s_t)$ is the expected, discounted, cumulative reward, taking action a_t in state s_t and following policy π_θ thereafter. It writes

$$Q^\pi(a_t, s_t) = r_t(a_t, s_t) + \mathbb{E}_\pi[\gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \tag{2.11}$$

where $r_t(a_t, s_t)$ is the reward that is obtained when taking action a_t in s_t and $\mathbb{E}_\pi[\gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$ is the expected reward from thereon following policy π_θ. $\mathbb{E}_\pi[\gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$ accumulates the reward over an infinite amount of steps which is intractable to compute. To bootstrap learning, we approximate the sum of expected rewards departing from state s_{t+1} by using the value function $V^\pi(s_t;\theta_v)$ that is parameterized by θ_v. Thus, Equation 2.11 becomes

$$Q^\pi(a_t, s_t) \approx r_t(a_t, s_t) + \gamma V^\pi(s_{t+1}). \tag{2.12}$$

In A3C, the network weights of policy π_θ and value function $V^\pi(s_t;\theta_v)$ are updated after $t_{\max}$ steps or sooner if the episode ends before. We empirically find that $t_{\max} = 5$ works well although it corresponds, at 30Hz, to a limited time span of 0.167sec. We can then extend Equation 2.12 by using the real reward of the first k time steps (bounded by $t_{\max}$), instead of only a single step, and approximate the rest using the value function $V^\pi(s_t;\theta_v)$:

$$Q^\pi(a_t, s_t) \approx \sum_{i=0}^{k-1} \left(\gamma^i r_{t+i}\right) + \gamma^k V^\pi(s_{t+k};\theta_v)\,, \quad k \leq t_{\max}. \tag{2.13}$$

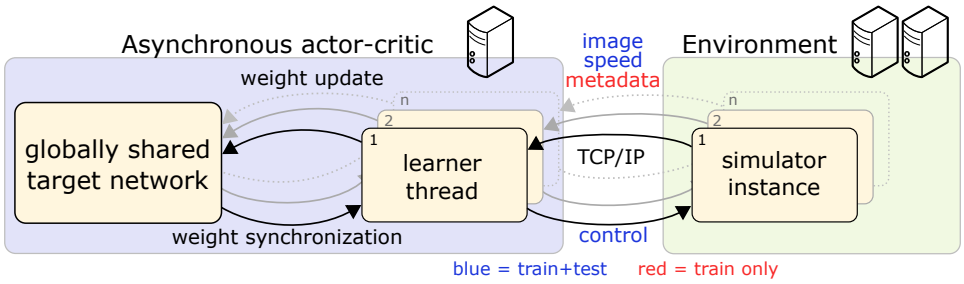


Figure 2.7: Scheme of our distributed training setup. Multiple simulator instances run on two machines (green box) and communicate through a dedicated API with the learner threads which run on a different machine (blue box). The learners update and synchronize their weights frequently with the shared target network in an asynchronous fashion.

By plugging the approximation of Q^π (cf. Equation 2.13) into Equation 2.10, we obtain an approximation of the advantage function

$$A(a_t, s_t; \theta, \theta_v) \approx \sum_{i=0}^{k-1} \left(\gamma^i r_{t+i}\right) + \gamma^k V^\pi(s_{t+k}; \theta_v) - V(s_t; \theta_v) \tag{2.14}$$

which can be used instead of the difference $R_t - b_t(s_t)$ to scale the policy gradient. Thus Equation 2.9 becomes

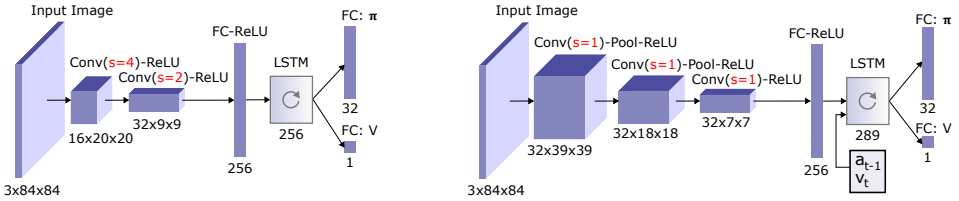
$$\nabla_\theta \log \pi_\theta(a_t|s_t;\theta) A(a_t, s_t; \theta, \theta_v) \tag{2.15}$$

which is the gradient estimate used in A3C to update the parameters of policy $\pi_\theta(a_t|s_t;\theta)$. Intuitively, the advantage function $A(a_t, s_t; \theta, \theta_v)$ measures whether the actions $a_t, a_{t+1}, ..., a_{t+k-1}$ were actually better or worse than expected.

The parameters of value function $V^\pi(s_t;\theta_v)$ are updated with an $\mathcal{L}_2$ loss between the estimated reward from the value function and the real reward.

For generality, it is said that the policy $\pi(a_t|s_t;\theta)$ (the actor) and the value function $V^\pi(s_t;\theta_v)$ (the critic) are estimated independently with two neural networks. In practice, both networks share all layers but the last fully-connected one as can be seen in Figure 2.8b.

Distributed learning. We choose A3C not only for its state-of-the-art performance, but also for its focus on asynchronous parameter updates that can originate from multiple learners. In our implementation, the learners are executed in separate threads to enable parallel learning. Our training setup can be seen in Figure 2.7. The asynchronous actor-critic (blue box) is run on a central PC where the learner threads can asynchronously update the weights of a globally shared network. The environment (green box) is run on other PCs (two additional PCs in our case). Each learner thread (agent)



(a) Mnih et al. (2016) (b) Ours

Figure 2.8: Comparison of the two used state encoders. (a) The CNN+LSTM architecture as proposed by Mnih et al. (2016) with a stride of four in the first convolutional layer and a stride of two in the second convolutional layer. (b) Our slightly deeper network consists of three convolutional layers with stride one. After each layer follows a 2x2 maxpooling operation. The LSTM takes the last action a_{t-1} and the current speed v_t as additional inputs.

communicates with its assigned instance of the simulator (WRC6 game) over LAN with an API created for this purpose using the TCP/IP protocol. In our case, we trained with nine agents (and nine game instances), uniformly distributed over three tracks. This multi-PC learning setup is necessary for our case, because WRC6 requires much more computation than comparable RL simulators, e.g. Atari games, and it would not be possible to run a high number of WRC6 instances on a single PC. Furthermore, this approach of parallelization allows training on different tracks (snow, mountain, coast) at the same time to foster generalization capabilities of the resulting policy.

2.4.2 Learning strategy

State encoder. The neural networks used in DRL are often extremely shallow, usually comprising only two or three convolutional layers (Mnih et al., 2016; Lillicrap et al., 2016), as compared to the deep networks commonly used in perception. For instance, the popular ResNet architecture (He et al., 2016) features versions with 18 to 152 layers. The reason why DRL limits itself to small architectures is twofold. First, training oftentimes already takes a long time, i.e. several days, and using a larger network would increase training time even further. Second, the simulators like Atari 2600 or TORCS (Wymann et al., 2000) are graphically very simple, so that small networks are sufficient to extract the necessary visual information. Even the road following task can in fact be achieved thanks to simple image features (road detection).

To encode the image, Mnih et al. (2016) use only two convolutional layers with strides 4 and 2 respectively, followed by one fully-connected layer. The resulting features are fed into a long short-term memory (LSTM) that can learn sequences of data. Using a recurrent network such as an LSTM is

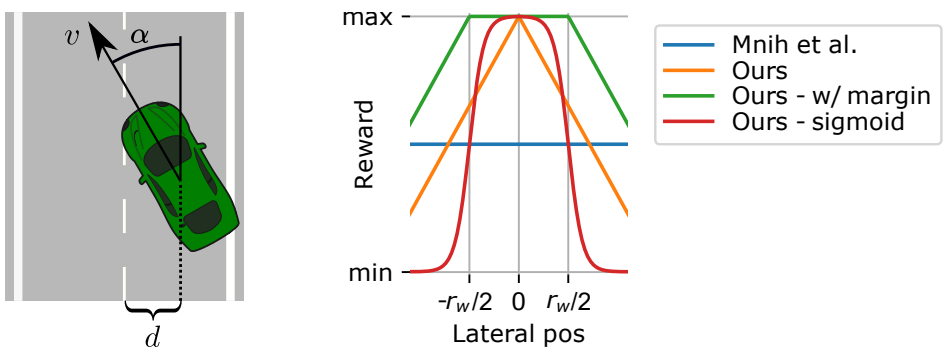
	control commands			
num. of classes	steering	throttle	brake	hand brake
27	$\{-1., -0.75, ..., 1.\}$	$\{0.0, 0.5, 1.0\}$	$\{0\}$	$\{0\}$
4	$\{-1., -0.5, 0.5, 1.\}$	$\{0.0\}$	$\{0\}$	$\{1\}$
1	$\{0.0\}$	$\{0.0\}$	$\{1\}$	$\{0\}$

Table 2.1: The 32 classes output for the policy network. First column indicate the number of classes with possible set of control values. Note the prominence of throttle commands.

beneficial for the driving task, because it can account for motion. The network of Mnih et al. (2016) is shown in Figure 2.8a.

In our network, depicted in Figure 2.8b, we add an additional convolutional layer as compared to Mnih et al. (2016), because WRC6 is visually more complex than TORCS, used by Mnih et al. (2016). Our network is a modified version of Kempka et al. (2016), using one-strided convolution and 2x2 max-pooling for downsampling as is common in perception networks. Our intuition behind one-strided convolution is that it should allow finer-grained vision and decision making. On top of the architecture from Kempka et al. (2016), we add an LSTM, which different from Mnih et al. (2016), takes the last action a_{t-1} and the current velocity v_t as additional inputs to facilitate temporal consistency in the network output. Note, that giving this additional information to the network does not amount to cheating as a human driver also has a speedometer and knows what the last action was.

Action space. Continuous control with Deep Reinforcement Learning (DRL) is possible (Lillicrap et al., 2016; Duan et al., 2016; Gu et al., 2016), but we use a discrete action space for simpler implementation of the loss function and action sampling. For the rally driving task, the agent needs to learn the control commands for steering $[-1, 1]$, throttle $[0, 1]$, brake $\{0, 1\}$ and hand brake $\{0, 1\}$. Note that the brake and hand brake commands are binary. Hand brake was added for the car to learn drifts in hairpin bends, since brake implies slowing down rather than drifting. The combination of all control commands at different step sizes yields 32 classes listed in Table 2.1. For example, one combination is $\{\text{steering} = 0.75, \text{throttle} = 0.5, \text{brake} = 0, \text{hand brake} = 0\}$. Although the total number of actions is arbitrary and usually plays little role in the final performance, two choices should be highlighted. First, hand brake is associated with different steering commands to favor drifting. Second, there is a majority of actions for acceleration to encourage speeding up (18 classes with throttle > 0 vs. 5 classes with brake or hand brake $= 1$). This prevents the problem of the agent never moving from its initial position, especially at the beginning of



(a) Metadata (b) Reward functions

Figure 2.9: Reward shaping. (a) In order to compute the reward, we use the following metadata that is provided at each time step by the simulator: the velocity of the car v, the angle α between the road direction and the car's heading, the distance from the center of the road d. (b) Different reward functions with respect to varying lateral position of the car. At 0, the car is at the track center and the road boarder is at $\pm r_w/2$. For this plot, we assume constant velocity.

the training. The static friction, genuinely modeled in WRC6, requires to apply the accelerator during some time before the car starts to move.

The probabilities of the 32 actions are predicted by the softmax output from the policy branch π of the network in Figure 2.8b.

Reward. The time needed to complete the track is the only "score" in race driving and is too sparse to be used as reward to learn vehicle control. Reward shaping addresses this problem by computing a reward at each time step with a reward function. Metadata, received at each time step from the simulator, can be leveraged to compute this frame-wise reward.

Lillicrap et al. (2016) and Mnih et al. (2016) use a reward function that projects the velocity of the car onto the direction of the road that writes:

$$R = v \cos \alpha \tag{2.16}$$

where α is the angle between the road and the direction of the car and v the car's velocity, as depicted in Figure 2.9a. We refer to this reward function as *Mnih et al.* In order to constrain the reward in the interval $[-1, 1]$, we normalize by the empirically observed maximum value, i.e. we divide the velocity v by 150 km/h. We clip the resulting reward between -1 and 1.

In empirical tests we have found that the reward function from Mnih et al. does not prevent the car from sliding along the guardrail. This makes sense since the guardrail follows the road angle. To address this issue we added the distance from the road center d, shown in Figure 2.9a, as a linearly

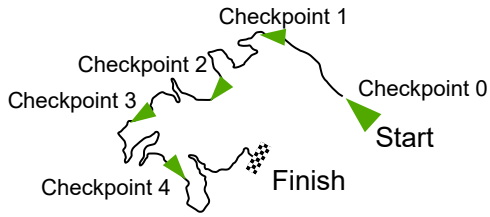


Figure 2.10: Instead of initializing the agents always at the beginning of the track, we propose to spawn them at different checkpoints to diversify training examples.

increasing penalty to Equation 2.16. In the following, we refer to this reward function as *Ours*:

$$R = v(\cos\alpha - |d|). \tag{2.17}$$

Note that we divide the distance d by 5 m for normalization purposes. Similarly to Lau (2016), our conclusion is that the distance penalty enables the agent to rapidly learn how to stay in the middle of the track.

Moreover, we propose two additional rewards that leverage the road width r_w. *Ours w/ margin* penalizes only with distance from track center when the car is off the road:

$$R = v(\cos\alpha - \max(|d| - 0.5r_w, 0)). \tag{2.18}$$

Ours sigmoid applies a smooth penalty as the agent deviates from road center:

$$R = v(\cos\alpha - \frac{1}{1 + e^{-4(|d| - 0.5r_w)}}). \tag{2.19}$$

We set the road width to $r_w = 3.5$m. A visualization of the four rewards as a function of the car's lateral position is displayed in Figure 2.9b, assuming a constant speed for simplification.

Ours w/ margin and *Ours sigmoid* penalize more when the car leaves the road and *Ours sigmoid* is smooth. Both of these reward designs are an attempt to decrease steering oscillations that can be seen in the video `https://youtu.be/e9jk-lBWFlw?t=15s` that we observed when using the reward *Ours*.

Agent initialization In previous DRL works in end-to-end driving (Mnih et al., 2016; Lau, 2016), the agents are always initialized at the beginning of the track. Such a strategy will lead to overfitting to the beginning of the training tracks. Instead, we choose to (re-)initialize the agents at random positions on the tracks after a crash or at training start. Due to technical constraints in WRC6, we are limited to five checkpoints (including start checkpoint) along the track as depicted in Figure 2.10. We think that it would be even better to initialize the agents at continuous locations (also

outside discrete checkpoints) to further increase diversity in experience. Ideally, also the lateral position and initial speed should be varied, but it was not possible in WRC6.

2.5 Experiments

This section describes our architecture setup and reports quantitative and qualitative performance in the World Rally Championship 6 (WRC6) racing game. Compared to the commonly used TORCS platform, WRC6 features a more realistic physics engine (grip, drift), graphics (illuminations, animations, etc.) and a variety of road shapes (sharp turns, slopes) as depicted in Figure 2.3. Additionally, the stochastic behavior of WRC6 makes each episode unique which is harder but closer to real conditions, i.e. two episodes never play out the same even when using a deterministic policy.

For better graph visualization all plots are shown as rolling mean and with standard deviation over 1000 steps. Qualitative driving performance is best seen in our video: https://youtu.be/AF0sryuSHdY?t=118.

2.5.1 Training setup

To speed up the pipeline and match the CNN input resolution, we disable several costly graphical effects and use a narrower field of view compared to the standard view in the game, as shown in Figure 2.6. The game's clock runs at 30 frames per second and the physical engine is on hold as it waits for the next action. Note that in practice, the simulator can be run faster than the simulated time and we reach approximately 100 frames per second during training, a speed-up of 3.33x. This means that in one day real-time, using 9 game instances, we can simulate ~ 30 driving days.

We use only front view images and speed for testing to allow for a fair comparison with a human driver. Likewise, images do not contain usual in-game info as done in RL with Atari games. Precisely, the head up display is disabled. This removes clutter, but also deprives the agent from information that a human player would have access to (percentage of the track completed, upcoming turns, etc.).

Although we think that initialization at random speeds would yield more diversity in training samples, we start agents (cars) always with velocity 0 km/h, because it is not possible otherwise in WRC6 due to technical limitations.

2.5.2 Metrics

To evaluate our approach, we define multiple new metrics that are independent from the reward and specific to the driving task. The metrics are

computed per episode. In our case, an episode starts at agent initialization at the beginning of the track or at a checkpoint and terminates when the agent reaches the finishing line or, more often, crashes, gets blocked or goes in the wrong direction (off road, wrong way); we refer to any of these terminal events as 'crash'. The metrics are defined as follows:

- **Covered distance**: Distance on the track outline that the agent progressed from the point of initialization to the point of crash or finish line.
- **Average speed**: Mean velocity obtained by dividing the covered distance by episode duration.
- **Collisions (car's hits per km)**: The number of collisions of the car with objects of the environment (guardrail, obstacles, etc.), divided by covered distance. We do not distinguish between violent and small hits.
- **Track exploration**: The percentage of the total track length that has been visited by the agents. For example, this metric improves when the agent learns to get past a difficult turn and can thus go farther, exploring unseen parts of the track.

While covered distance and track exploration are indicators during training for how fast the agents learn to master the track, average speed and collisions can be used to distinguish driving styles.

2.5.3 Performance evaluation

Plots in Figure 2.11 show mean (dark) and standard deviation (light) for the three training tracks over 140 million steps. During training, the agents progress along the three very different tracks simultaneously. Overall, the agents successfully learned to drive despite the challenging track appearances and physics at an average speed of 72.88km/h and cover an average distance of 0.72km per run. We observe a high standard deviation for the covered distance as it depends on the difficulty of the track part where the car is initialized.

The bottom plot reports collisions (car's hits per km). Despite not being explicitly penalized in the reward function, the hits decrease as training progresses, probably because collisions lead to a speed decrease. After training, the car hits scene objects 5.44 times per kilometer. Note that such collision behavior is impractical for later adaptation to normal driving. However, our reward is still appropriate for racing, favoring fast over cautious driving.

Crash locations, binned into five meter segments along the track, are colored from black to yellow in Figure 2.12. The latter highlights difficult

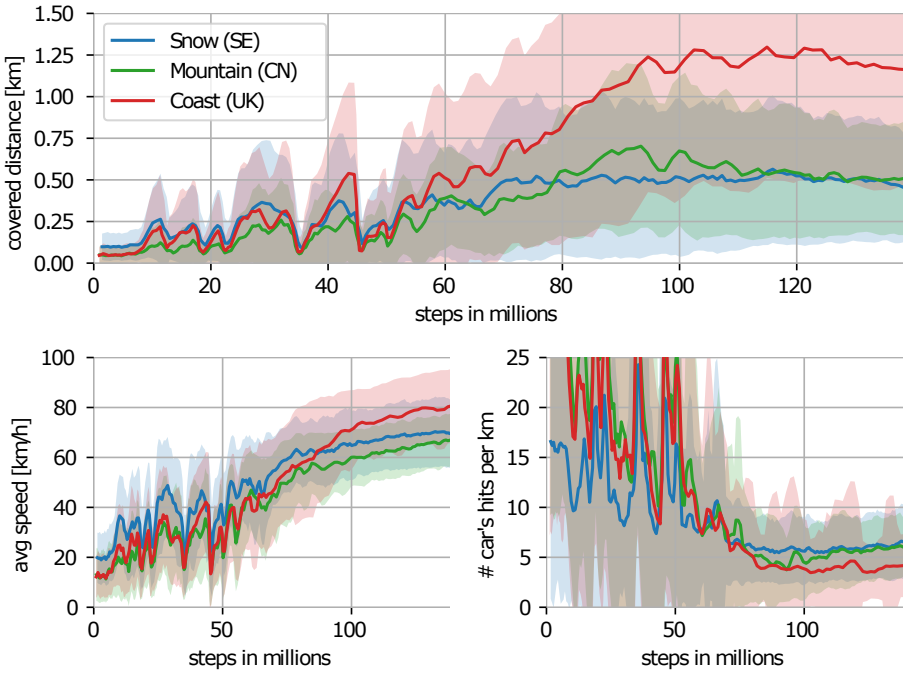


Figure 2.11: Training performance reported as rolling mean (dark) with standard deviation (light). The agent had more difficulty to progress on the *mountain* and *snow* tracks as they exhibit a greater number of sharp turns and the snow track has a slippery road.

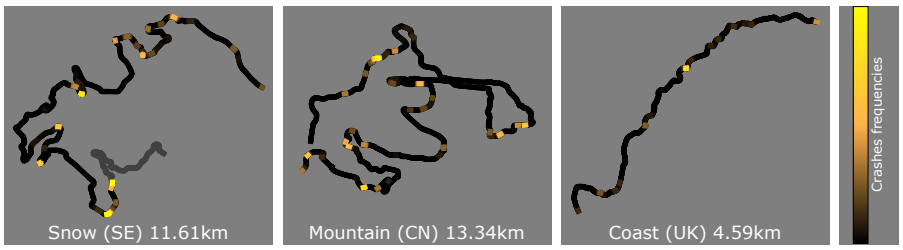


Figure 2.12: Crash frequencies for the three training tracks. Crashes, drawn in yellow, are usually concentrated at difficult locations such as turns. There is also an unexplored part at the end of the challenging Snow track (in gray). Some difficult hairpin bends are mastered with no or few crashes by the learned policy.

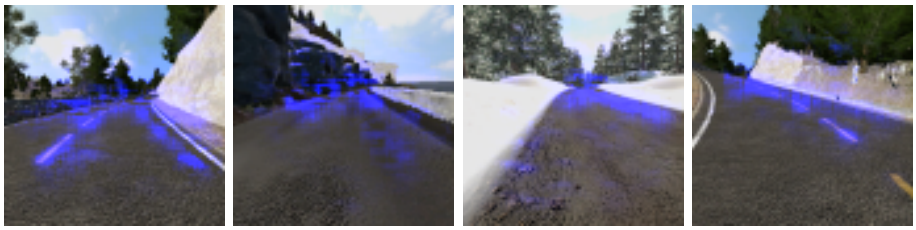


Figure 2.13: Visualization of back-propagation where positive gradients for the chosen actions are highlighted in blue. Despite various scenes and road appearances the network learned to detect road edges or lane markings and relies on them for control.

locations on the tracks, but also shows that the agent can even pass some hairpin bends.

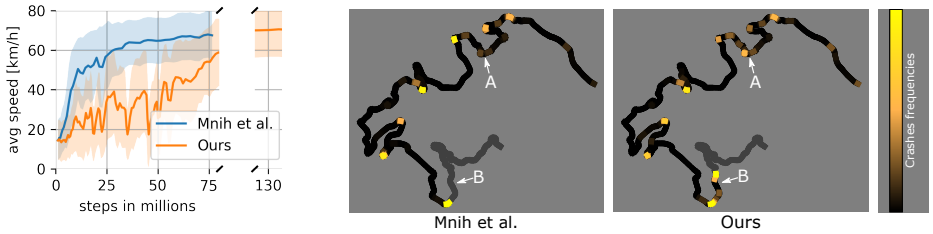
From a qualitative point of view, the agent drives rather smoothly and even learned how to drift with the hand-brake control strategy. However, the policy does not achieve optimal trajectories from a racing aspect (e.g. taking turns on the inside) which is possibly because the car will always try to remain in the track center because of the deviation penalty in our reward function. For the snow track, where the road is very slippery, we observe that the car often diverts from the road when it goes too fast. Although on all tracks the average number of hits is relatively high, the context of a racing game is very complex and we found that even the best human players collide with objects such as the guardrail.

To visualize how the input image influenced the driving decisions of the neural network, Figure 2.13 shows guided back propagation (Springenberg et al., 2015) for several scenarios. The visualization is computed as follows: given an action, positive gradients are backpropagated through network until the input space. It thus highlights regions in the image that have been important for the actual control prediction. Despite the various scene appearances the agent consistently uses the road edges and curvature as a strong indicator. This makes sense, as it helps to follow the road.

2.5.4 Ablation studies

In the following we evaluate our design choices of state encoder, reward function and agent initialization strategy.

State encoder. Figure 2.14 compares the performance of our CNN against the smaller network from Mnih et al. (2016), shown in Figure 2.8a. Looking at the average speed in Figure 2.14a, the performance is similar for both networks, but the smaller network from Mnih et al. (2016) converges faster than for ours (80 versus 130 million steps). Both networks also lead to similar exploration of the track, as depicted in Figure 2.14b. The crash locations



(a) Performance (b) Locations of episode terminations

Figure 2.14: Evaluation of our state encoder versus the encoder of Mnih et al. (2016). (a) The small CNN from Mnih et al. (2016) (blue) converges faster than ours (orange). (b) Locations where crashes occur. Both networks are comparable, but our network has slightly more crashes at location A and explores a little more of the track at locations B.

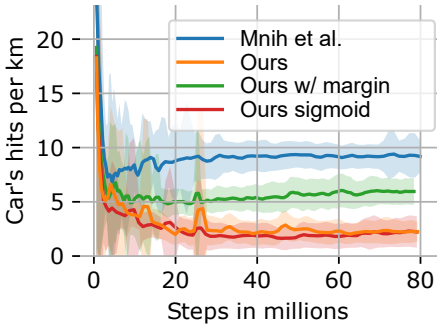
(highlighted in yellow) are similar even on such a difficult track, except in the region labeled as section A where Mnih et al. (2016) crashes less often and section B only explored by our network.

Despite the longer convergence, the performance of our network is better compared to the smaller network with $+89.9m$ average covered distance ($+14.3\%$) and -0.8 average car's hit per kilometer (-13.0%). Such analysis advocates that our network performs better, but at the cost of doubling training time. In light of these results, we argue that there is little interest in using our architecture, at least with the simple road following task.

Reward functions. In the following, we compare the reward function of Mnih et al. (2016) of Equation 2.16 against the three variants of our reward function of Equations 2.17, 2.18 and 2.19, referred to as *Ours*, *Ours w/ margin* and *Ours sigmoid*, respectively. For this experiment, we use the network architecture of Mnih et al. (2016) and train on a single track to minimize variance and enhance interpretability.

Figure 2.15a shows the collisions (hits/km, lower is better) during the training, and report the average speed and collisions in Table 2.15b. Our three rewards using the distance from track center lead to a significant drop in hits, precisely *Mnih et al.* 9.26hits/km versus *Ours sigmoid* 2.24hits/km. Out of our three rewards, *Ours w/ margin* performs worst in terms of collisions, probably because it does not teach the agent to center on the road, as the reward function is piecewise constant within the road width, as depicted in Figure 2.9b. Furthermore, we observe that fewer collisions come at the cost of lower speed: *Mnih et al.* 106.9km/h versus *Ours sigmoid* 89.8km/h. In conclusion, the reward from Mnih et al. (2016) leads to faster, but rougher driving.

Note that we observed faster convergence with our rewards than with



(a) Training performance

	speed [km/h]	collisions [hits/km]
Mnih et al.	**106.9**	9.26
Ours	91.4	2.25
Ours w/ margin	100.3	5.95
Ours sigmoid	89.8	**2.24**

(b) Mean values

Figure 2.15: Quantitative comparison of different reward functions.
(a) Training curves using collisions (hits per km) as performance metric. Our three proposed reward function lead to less collisions as compared to Mnih et al.
(b) Mean over all training episodes for speed (higher is better) and number of collisions (lower is better). Our reward functions lead to less crashes at the expense of a decrease in speed.

the reward from Mnih et al. (2016), as they bootstrap learning by explicitly teaching the agent to center laterally on the road with distance penalty d. We qualitatively observed less oscillations with *Ours sigmoid*; we hypothesize that this is because of its smoothness around the track center as shown in Figure 2.9b. As *Ours sigmoid* was implemented at the very end of this work, we use *Ours* in the rest of the experiments.

Agent initialization. To compare the performance of the *start* and *checkpoint* initialization strategies, we carry out two trainings from scratch. In the first training we use our random initialization at different checkpoints and in the second one initialization at the beginning of the track.

When the agents start at different positions, *covered distance* (as defined in Section 2.5.2) is not a valid metric during training, because an agent starting half way through the track can not cover more than 50% of the track length[2]. Instead, we use *track exploration* as metric. In Figure 2.16, our random checkpoint strategy (plain lines) exhibits a significantly better exploration than the usual start strategy (dashed lines). For the easiest coast track (green), both strategies reach full exploration of the track, but random checkpoint initialization converges faster. For complex snow (blue) and mountain (green) tracks, our strategy improves the exploration by large margins, +32.20% and +65.19%, respectively.

The improvement due to our strategy is easily explained as it makes full use of asynchronous learning of A3C. A wide variety of track sections can be

[2]The tracks in WRC6 do not loop.

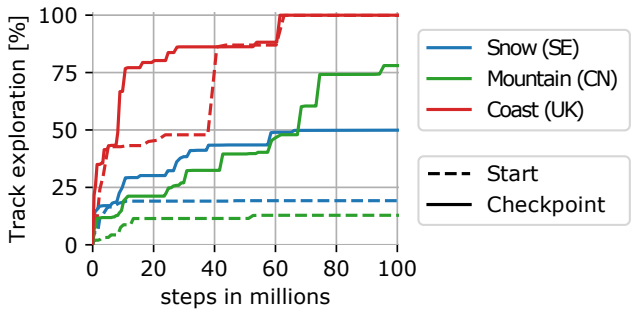


Figure 2.16: Comparison of training performance with different initialization methods. The agents are spawned at the *start* (dashed line) of the track or at *random checkpoints* (plain line) as depicted in Figure 2.10. We use the metric of track exploration which corresponds to the percentage of track seen by the agent. The experiment is run on three different tracks: Snow (SE), Mountain (CN) and Coast (UK).

explored in parallel at the same time which benefits experience decorrelation.

Figure 2.16 shows that track exploration increases in a non-linear fashion with sudden jumps. Indeed, some track segments (e.g. sharp turns) are particularly difficult to pass and require many attempts before the policy succeeds, leading to a segment-by-segment training progress.

2.5.5 Generalization

WRC6 is somewhat stochastic (physics, object animations, illumination changes, etc.). Therefore, the performance reported on the training tracks already includes some generalization to slight perturbations in the environment. Still, we want to answer the following questions: Can the agent drive on unseen tracks? Can it drive respecting the speed limit? How does it perform on real images?

Unseen tracks. We tested the trained policy on tracks in WRC6 with different road layout. The agent could successfully follow the road in those test scenarios. This shows that the network incorporated general driving concepts rather than learned a track by heart. Qualitative performance on an unseen test track in the mountain region is shown in video `https://youtu.be/AF0sryuSHdY?t=131`.

Racing VS Normal driving. As the reward favors speed without direct penalization of collisions, the agent sometimes collides with objects such as the guardrail. This can occasionally avoid a slowdown of the car as it "bounces" off the guardrail, but is inappropriate for normal driving. In the following, we evaluate how our policy could be transposed to normal

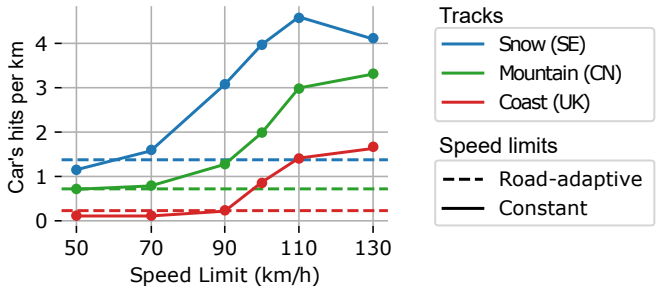


Figure 2.17: Influence on the number of collisions (hits per km) of imposing segment-wise road-adaptive (dashed lines) and whole track constant speed limits (plain lines) on the three training tracks.

Figure 2.18: Prediction of longitudinal and lateral commands on real videos. Note that the agent can handle situations never encountered (other road users, multi-lanes).

driving by imposing speed limits. We implement it such that if the car goes faster than the speed limit, we eliminate the actions containing throttle ≥ 0 from the predictions and select the action with maximum probability among the remaining ones. We test different *constant* whole track speed limits (50 km/h, 70 km/h, etc.), depicted as continuous lines in Figure 2.17, and *road-adaptive* speed limit computed individually for each road segment[3], shown as dashed lines in Figure 2.17.

As one could expect, the number of collisions significantly decreases when imposing lower speeds. Even though this shows that test time speed limits help to reduce collisions as compared to rally driving, this is by far not sufficient to achieve normal driving. An alternative approach could be to re-train with a reward function that directly incorporates speed limits.

[3]In order to compute our road-adaptive speed limit, we used the so-called design speed which is used to decide the speed limit on real road infrastructure. It is computed as a function of local curvature and superelevation using American infrastructure standards (Hancock and Wright, 2001).

Real videos. Finally, we tested our agent on real videos, in open-loop, i.e. predicting the control actions without applying them. We selected publicly available dash cam videos of country roads that feature similar viewpoints to WRC6[4,5,6], cropped and resized. Figure 2.18 shows the control output of our policy and guided backpropagation of three frames. Results on sequences can be seen in our video https://youtu.be/AF0sryuSHdY?t=154. Although the results are preliminary as we cannot act on the video (i.e. control commands are never applied), the decision performance, such as the road following and avoidance of oncoming traffic, is acceptable for such a shallow network.

Note that these experiments are limited, because as they are conducted only qualitatively in open loop, i.e. the possibly bad control commands are never applied in the video. There is still a large gap between predicting reasonable frame-wise actions and reliable control over a whole sequence with recovery from difficult situations, e.g. stable lateral control without oscillation or re-centering in the lane after deviation. The ideal way to evaluate our model would have been in a real car. However, this is costly and difficult from a safety perspective. Simpler options for evaluation that could be considered in future research are the following. First, conduct open loop testing on a dataset where the real action (steering angle, throttle, etc.) has been registered (e.g. Udacity[7]), to compare quantitatively between a human driver and the model's predictions. Second, and more interesting, carry out a closed loop experiment in another simulator such as CARLA (Dosovitskiy et al., 2017) or GTA V[8].

2.6 Discussion

Choice of WRC6. WRC6 certainly has better graphics and physics than TORCS and even more recent CARLA. However, there are some major drawbracks to it. Working with a game such as WRC6 was complex and required a significant amount of engineering work. As it is proprietary it is not possible to make changes to the game engine directly[9]. Moreover, because it was designed as a video game, it lacks functionalities that are important to an RL research platform such as adjusting the game clock (frequency at which frames are produced), running multiple agents in the same environment, initializing agents at arbitrary locations, etc. We thus advise to use research-oriented driving simulators, among which CARLA (Dosovit-

[4]USA mountain dash cam: https://youtu.be/52vaq3dqTk4?t=3437

[5]Croatia mountain dash cam: https://youtu.be/HjNxDC5ftG8?t=16s

[6]India tea fields dash cam: https://youtu.be/KCVhK8QJujk?t=59s

[7]https://github.com/udacity/self-driving-car

[8]https://github.com/aitorzip/DeepGTAV

[9]Valeo paid the game developer Kylotonn to design an API to access their game for the purpose of a Deep Learning demo at the Consumer Electronics Show 2017. This API was then used to conduct further research.

skiy et al., 2017) – released posterior to our work – appears to be the best option and is largely used in the community today. Note that CARLA was specifically designed for research, and features complex urban driving with traffic lights and other road users (pedestrians, bikes and cars). Other interesting simulators include AirSim (Shah et al., 2018) and GTA V. However, AirSim misses moving objects and GTA V is a commercial video game and research usage relies on a plugin[8]. Thus, the proprietary game itself can not be changed and adapted to research purposes, e.g. it does not support multiple agents.

Choice of A3C. A3C (Mnih et al., 2016) facilitated the implementation of distributed learning and thereby sped up our experiments. Still, training time was long with around 4 to 5 days on 3 machines even though we disabled high quality image rendering, a limitation of our approach. The more recent actor-critic algorithm PPO (Schulman et al., 2017) should be preferred to A3C as it has better performance and more stable training than A3C. However, the drawback of actor-critics is that they are 'on-policy': new trajectories need to be sampled after each policy update, while old trajectories are discarded. This is acceptable when simulation time is negligible compared to the time needed to update the parameters of the model through backpropagation. This is typically the case in simple Atari games, but less so for complex WRC6. To improve data efficiency, off-policy value-based algorithms such as Rainbow (Hessel et al., 2018) should be investigated, where trajectories are stored in a replay memory and can be re-used several times for training instead of being discarded immediately after the first parameter update.

2.7 Conclusion

In this chapter we used the deep reinforcement learning (DRL) method A3C to learn a race driving policy in the video game WRC6. Therefore, we adapted the network architecture, action space, reward function and agent initialization strategy to our specific needs and implemented distributed learning. Also thanks to significant engineering efforts in our work, we were, to the best of our knowledge, the first to demonstrate that DRL can be used in complex realistic simulators to learn full control of the car.

We achieved fast rally driving with our approach on three race tracks shown in the videos. Moreover, we introduced new metrics to thoroughly evaluate and justify our design choices. Our policy was trained in parallel on three different tracks and shows some degree of generalization to other tracks in the game and open-loop action prediction on real driving videos. While generalization was lightly evaluated, the preliminary results are promising especially given the large domain gap between the synthetic and real data.

In this chapter, we tackled the whole chain of autonomous driving from raw data input to vehicle control. This work is only weakly related to the core of this thesis that focuses on 2D-3D scene understanding as we only used the image as input and no 3D point cloud. DRL training is very time-consuming and we have observed that using only slightly larger networks to encode the images drastically increases training time even more. Therefore, it is highly impractical to study 2D-3D scene understanding in the RL setting and we will focus solely on the perception part in the remaining chapters.

Complétion de cartes de profondeur éparses via fusion avec RGB dense

French Summary of the Chapter "Fusing sparse depth and dense RGB for depth completion"

Dans ce chapitre, nous projetons des nuages de points LiDAR 3D dans l'espace image 2D, résultant en des cartes de profondeur éparses. Nous proposons une nouvelle architecture encodeur-décodeur qui fusionne les informations de l'image et la profondeur pour la tâche de complétion de carte de profondeur, améliorant ainsi la résolution du nuage de points projeté dans l'espace image.

Chapter 3

Fusing sparse depth and dense RGB for depth completion

The contributions of this chapter were published in (Jaritz et al., 2018b):

Jaritz, M., de Charette, R., Wirbel, E., Perrotton, X., and Nashashibi, F. (2018b). Sparse and dense data with CNNs: depth completion and semantic segmentation. In *3DV 2018*.

Contents

3.1 Introduction **46**
3.2 Related Work **49**
3.3 Method **54**
3.3.1 Network Architecture 54
3.3.2 Sparse Data Training 56
3.3.3 Analysis of Validity Mask 58
3.4 Experiments **59**
3.4.1 Datasets 59
3.4.2 Implementation 60
3.4.3 Depth completion 60
3.4.4 Semantic Segmentation 65
3.5 Discussion **69**
3.6 Conclusion **72**

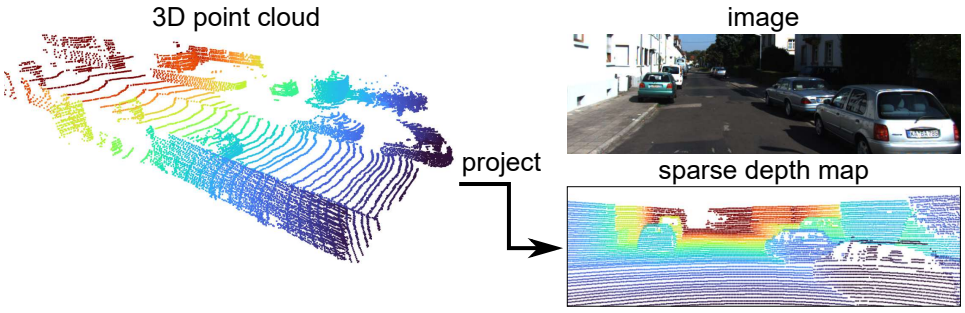


Figure 3.1: Projection of a 3D LiDAR point cloud into a camera image space. The coloring is based on the depth. Data from Kitti (Geiger et al., 2013).

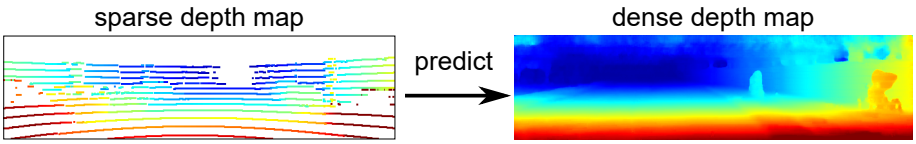
3.1 Introduction

In Chapter 2 it was shown that it is possible to train a neural network that takes raw sensor data as input and directly outputs the vehicle control commands. From this chapter on, we study multi-modal inputs and for simplicity exclusively focus on perception, rather than the whole end-to-end pipeline. In particular, we investigate the fusion of camera and LiDAR which have much higher resolution than radar and ultrasonic sensors. While LiDAR actively measures with laser beams producing a sparse 3D point cloud, camera passively captures the scene using external light sources yielding a dense 2D image. The complementarity of sparse but precise 3D geometry information from LiDAR and the dense visual appearance information from camera makes them good candidates for fusion.

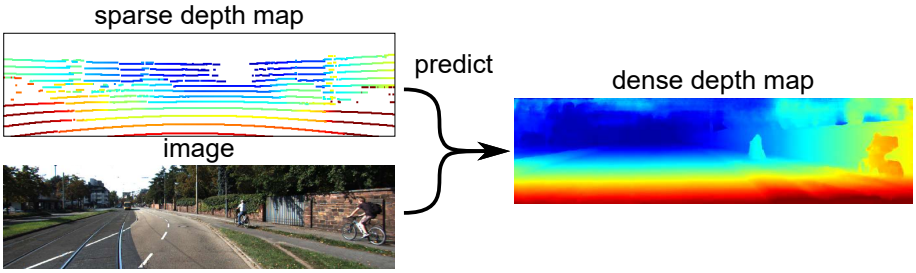
In order to combine 2D-3D data, a joint representation space is needed. A practical choice, because it allows for the usage of off-the-shelf 2D CNNs, is to project all 3D points into the 2D camera image space using extrinsic and intrinsic calibration. A sparse depth map can be obtained by assigning the corresponding depth value to each projected 2D pixel (see Figure 3.1).

Once the 3D data has been projected into the 2D camera image space, the task of depth completion suggests itself. It consists in filling the holes in a sparse depth map (see Figure 3.2a) in order to densify the sparse 3D data. This is similar to image inpainting which, before the arrival of deep learning, was achieved through sophisticated interpolation of valid data, for instance via patch-based synthesis (Efros and Leung, 1999) or sparse dictionary learning (Mairal et al., 2008), though failing at completing large holes. Deep learning on the other hand, can complete large chunks of missing data from learned appearance priors (Yu et al., 2018).

Uhrig et al. (2017) have introduced the depth completion task on the Kitti dataset (Geiger et al., 2013). The sparse input depth map comes from a LiDAR point cloud that is projected into the camera frame (see



(a) sparse depth input



(b) fusion of sparse depth and RGB input

Figure 3.2: Depth completion is the task of filling the empty holes in the sparse input. It is possible to predict the dense depth map only based on sparse depth input (a) or to fuse with the dense image (b). Data from Kitti (Geiger et al., 2013).

Figure 3.1). The dense ground truth is obtained by accumulating 11 LiDAR frames through projection of the 5 preceding and 5 following LiDAR frames into the current one. Then a comparison with the disparity from stereo camera is performed and inconsistent depth pixels from accumulated LiDAR are eliminated. Hence, depth completion is a task where ground truth can be gathered in an inexpensive, automated manner.

There are multiple challenges in depth completion. The most important is, that the input depth map is sparse. As 2D CNNs have been designed and optimized for dense images, their application to sparse data requires adaptation. For instance, due to sparsity, object boundaries are not clearly visible and the sparsity is heterogeneous depending on the location in the image and the distance of the objects. In order to guide the depth completion with RGB data from the dense camera image, an effective fusion mechanism is necessary, where the high resolution and visual appearance context of the image can be leveraged to support the dense depth prediction, as depicted in Figure 3.2b. Besides the question of a suitable architecture, test time robustness to changes in the density (e.g. lower LiDAR resolution), is critical.

In order to address these challenges, we propose a new encoder-decoder architecture, a camera image + sparse depth (RGB+sD) fusion strategy and a sparse training scheme.

We derive a U-Net like architecture (Ronneberger et al., 2015) as used in semantic segmentation, that allows us to incorporate context aware depth

predictions due to a large receptive field, which is particularly helpful in low-density regions. We employ the powerful, but efficient NASNet encoder to learn effective features and design a decoder with transposed convolutions for feature map upsampling that finally produces a dense depth map prediction of same size as the image. Skip connections between encoder and decoder allow the preservation of details.

For RGB+sD fusion, we propose an early fusion strategy where we use separate encoders for each modality. RGB and sD features are then concatenated and fed into a joint depth decoder. The incorporation of dense RGB data allows us to recover sharp object boundaries and make a better informed depth completion.

Furthermore, we present a sparse training scheme, where we vary the input sparsity during training in order to guarantee test-time robustness. This can be seen as an alternative to sparse convolution as proposed by Uhrig et al. (2017) who employ validity masks (binary masks that indicate available and unavailable pixels). We analyze validity masks and experiment with varying sparsity using synthetic and real data.

Finally, we extend to the task of semantic segmentation to validate our architecture and investigate if dense segmentation can be predicted from sparse depth and how RGB+sD fusion can improve performance.

Our contributions can be summarized as follows:

- We design an encoder-decoder architecture that is well-adapted to the depth completion task.
- We introduce an RGB+sD late-fusion strategy to effectively combine information from the complementary modalities of sparse depth and RGB.
- We analyze how CNNs behave at different densities in synthetic (random sampling) and real data (varying number of LiDAR layers) and find that best performance is obtained when a network is specialized in a certain density, that a validity mask is not needed and that robustness at test time can be achieved through data augmentation during the training.
- We extend to the task of semantic segmentation.

We achieve state-of-the-art results on the Kitti Depth Completion benchmark at the time of submission, and showcase the benefit of sparse depth + RGB fusion for semantic segmentation. Since the publication of our work (Jaritz et al., 2018b), numerous works with improved scores have been submitted to the Kitti depth completion benchmark. We chose to provide the reader with a complete review of the literature but separate posterior work to our contributions for clarity.

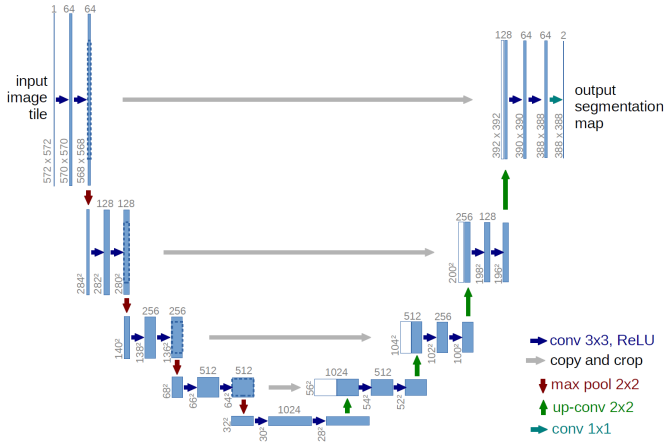


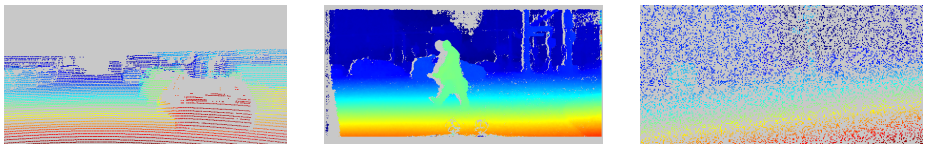
Figure 3.3: The U-Net architecture (Ronneberger et al., 2015). The architecture consists of an encoder (downsampling part) and a decoder (upsampling part). Skip connections between encoder and decoder are realized by copying and concatenating the downsampled encoder features to the upsampled decoder features. Source: Ronneberger et al. (2015).

3.2 Related Work

Our contribution mainly answers the question of how to predict a dense depth map of same spatial resolution as the input – by fusing a sparse depth map and a dense image with a neural network. Therefore, we divide the related works by input type: dense, sparse and dense + sparse. The latter focuses on recent works in depth completion that have been published since our contribution discussed in this chapter.

Dense Inputs. While depth completion is a regression and semantic segmentation a classification problem, they are both pixel-level predictions with similar spatial structure. More precisely, for both tasks the output is of equal spatial resolution (or aspect ratio) as the input, which is why both tasks share similar architectures.

For semantic segmentation, Long et al. (2015a) proposed Fully Convolutional Networks (FCNs) where they remove feature map reshaping and fully connected layers from classification networks and instead upsample lower-resolution features after the third and fourth convolutional layer using bilinear upsampling or transposed convolution to obtain a high resolution output. Ronneberger et al. (2015) and Badrinarayanan et al. (2017) introduce deeper architectures with mirrored encoder (CNN backbone with downsampling) and decoder (upsampling). The U-Net architecture (Ronneberger et al., 2015) is depicted in Figure 3.3, where skip connections convey fine-grained details from the encoder to the decoder. PSPNet (Zhao et al., 2017) uses



(a) Structured (b) Patches (c) Uniform

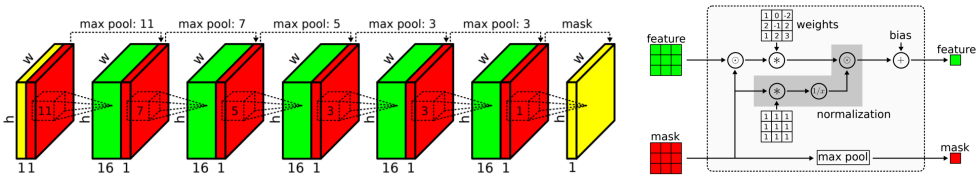
Figure 3.4: Different sparsity patterns from sensors such as LiDAR (a), stereo camera (b) or synthetic data (c).

Pyramid Pooling, where the multi-scale feature maps are upsampled to the same resolution and concatenated. An alternative to downsampling feature maps is to dilate the convolutional kernels (Chen et al., 2018). In this way, skip connections can be omitted as the same resolution is kept throughout the network.

For depth estimation from a single RGB image, priors on object and scenes are learned, for example with VGG (Eigen et al., 2014) or deeper ResNet-50 (Laina et al., 2016) networks. Kuznietsov et al. (2017) learn to predict depth in a supervised manner with sparse LiDAR measurements. Godard et al. (2017) leverage self-supervised learning with stereo camera data by using the predicted depth to warp the left camera image to the right camera image and then compute a loss between them.

Sparse Inputs. The nature of sparse data varies with the sensor and scenario, as can be seen in Figure 3.4. For instance, LiDAR data exhibit *structured sparsity* due to the scanning behavior with discrete angles in spherical coordinates (Figure 3.4a). Stereo vision or structured light sensors deliver dense data with *patches* of missing data (Figure 3.4b). Data can also be sparsified artificially, commonly through *uniform* sampling (Figure 3.4c). Classically, sparse 2D inputs are considered for inpainting of *uniform* sparsity using local interpolation (Ku et al., 2018) or guided optimization (Silberman et al., 2012) for *patches* sparsity.

As CNNs are designed to operate on dense data, a common strategy is to transform sparse data to a 2D (Uhrig et al., 2017; Ren et al., 2018) or 3D (Riegler et al., 2017) grid with *holes*. A validity mask can be given as additional input to express valid or missing data (Uhrig et al., 2017; Ren et al., 2018). Uhrig et al. (2017) – who established the groundwork of CNN based sparse depth completion – propose sparsity invariant convolution to tackle the problem (see Figure 3.5b). It consists of a normal dense convolution, but the result is then normalized by the number of valid pixels as stored in the validity mask. However, it can blur the predicted output due to dilation of the validity mask, as discussed in Section 3.3.3. Also, the network might be too small to learn effective features and incorporate context (see Figure 3.5a).



(a) Network (b) Sparse Convolution

Figure 3.5: CNN based depth completion with sparse convolution proposed by Uhrig et al. (2017). (a) The network is rather shallow with 5 convolutions and small receptive field as there is no downsampling. Along with the sparse depth, a validity mask (red) is provided as input to the CNN. (b) Definition of sparse convolution, where the dense convolution result is normalized using the validity mask. Source: Uhrig et al. (2017).

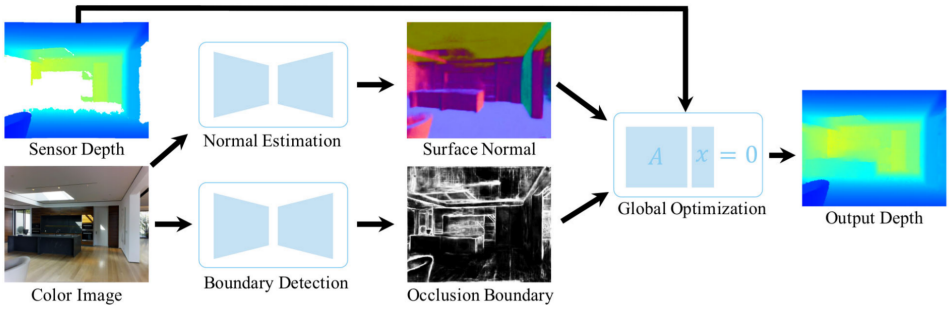


Figure 3.6: Zhang and Funkhouser (2018) predict surface normals and occlusions boundaries from RGB image only. Then they use the predictions and the sparse sensor depth jointly in a global optimization to obtain the dense depth. Source: Zhang and Funkhouser (2018).

If the aim is to reduce computation, a validity mask is necessary to define where convolution should take place (valid locations). Whereas Ren et al. (2018) define valid blocks, Graham and van der Maaten (2017) use hash Tables for pixel-wise indexing. A different approach to handle sparse data, surpassing grids altogether, is to define order-invariant operations on lists, such as a 3D point cloud (Qi et al., 2017a). These methods specifically focus on sparse data, but unfortunately their output is also sparse which makes them unsuitable for depth completion, where the goal is to predict a dense output.

Sparse + Dense Inputs. The problem of fusing dense and sparse data was little addressed at the time of our contribution. Zhang and Funkhouser (2018) predict surface normals and occlusion boundaries from dense RGB which are then used in a non-learned global optimization scheme to com-

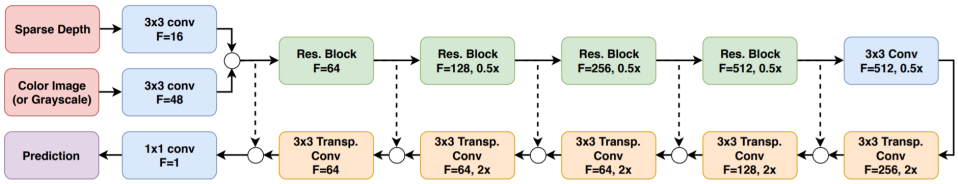


Figure 3.7: Ma et al. (2019) use early fusion, where sparse depth and image and concatenated after the first convolution. The encoder is a ResNet backbone and skip connections between encoder and decoder are depicted by dashed lines. Source: Ma et al. (2019).

plete the sparse depth input (see Figure 3.6). Since our contribution, depth completion has become a hot research topic, sparking many new approaches that improve over our score on the Kitti benchmark (Uhrig et al., 2017).

Our contributions (Jaritz et al., 2018b), presented in this chapter, include fusing sparse depth and dense RGB with a late-fusion strategy. Our architecture is inspired by U-Net (Ronneberger et al., 2015) with skip connections, but using an individual encoder for each modality (RGB and sparse depth) and a common decoder to predict the dense depth.

As expected, all recent works fuse sparse depth and dense RGB which is logical as dense RGB data can efficiently guide completion of missing data in the sparse input depth map. Yang et al. (2019) chose to use our late-fusion architecture, while replacing our NASNet encoder blocks (Zoph et al., 2018) by ResNet blocks which is a good choice, as it simplifies implementation and does not hurt performance. Eldesokey et al. (2019) similarly show that our late-fusion approach leads to better performance than early-fusion where RGB and sparse depth are concatenated in the beginning. Both works experiment with confidence prediction in order to improve performance. Ma and Karaman (2018) simply concatenate RGB and sparse depth, resulting in a 4-channel tensor that is input into an encoder-decoder network. In more recent work, Ma et al. (2019) also use a ResNet based encoder-decoder with early-fusion (see Figure 3.7). While they achieve good performance with self-supervised learning using a photometric loss, obtained through warping the right to the left stereo camera image using the predicted depth map, it does not help to outperform supervised learning, even when supervised and self-supervised losses are combined. Ma et al. (2019) also present a purely supervised setup and can increase performance by optimizing design choices such as clipping the output depth at a user-defined threshold, using a different loss function, employing skip connections and by removing downsampling to retain fine-grained details. Van Gansbeke et al. (2019) present an approach with a local and global network branch. The outputs of both branches are weighted by a confidence prediction and then summed. Xu et al. (2019) use the same ResNet based early-fusion U-Net as Ma et al.

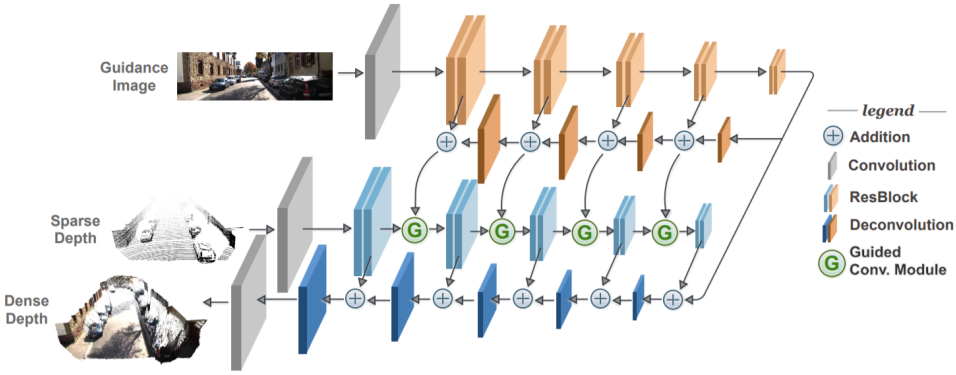


Figure 3.8: Tang et al. (2019) use a similar U-Net like architecture and independent encoders for RGB and sparse depth like in our work (Jaritz et al., 2018b), but introduce an additional decoder for RGB guidance (in dark orange), which features are then used in the sparse depth encoder leading to a sequential processing of RGB and sparse depth. Source: Tang et al. (2019).

(2019) to predict coarse depth, surface normals and a confidence map. Then, the three predictions are fused in an iterative approach to obtain the final refined depth. Qiu et al. (2019) modify our late-fusion U-Net by using summation instead of concatenation for the skip connections which allows to reduce the number of parameters. Furthermore, they estimate dense depth and surface normals in a 2-branch architecture. Both branch outputs are combined to obtain the final result. Tang et al. (2019) is the current state-of-the-art and use a similar architecture with late-fusion as us, but add RGB guidance in the depth encoder (see Figure 3.8).

All the preceding works presented so far, project the 3D point cloud into the 2D camera space and then input it into a standard 2D CNN. An important problem due to projection is that two points can be neighbors in 2D, although they are far apart in 3D, which occurs especially at object boundaries. Chen et al. (2019b) address this problem by building a 2D-3D fusion block where a dense 2D convolution is computed in parallel to a continuous convolution in 3D space. For the latter, the neighboring points are sampled with kNN in 3D space for each pixel, which effectively excludes 2D neighbors that are far away in 3D. After that, the two resulting feature maps are merged together again. By stacking multiple of those 2D-3D fusion blocks, a deep network is created.

In summary, while recent methods outperform ours, they re-use several of our findings and confirm our design choice of fusing sparse depth and dense RGB inputs with a U-Net based architecture.

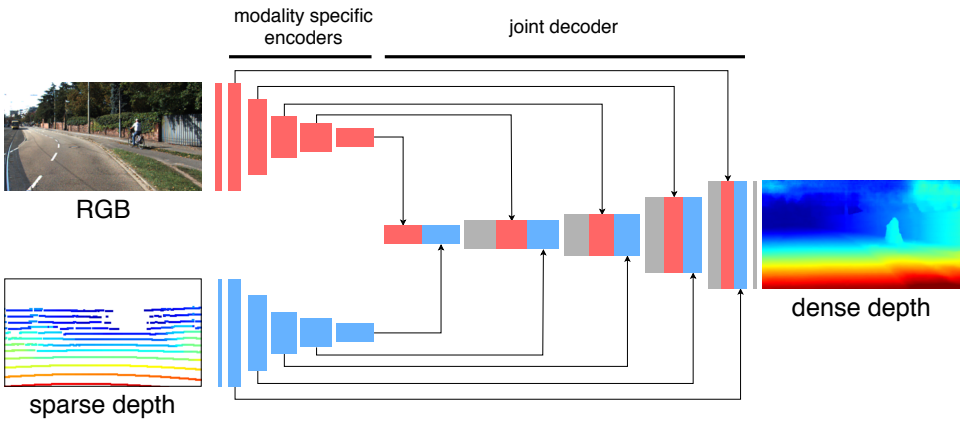


Figure 3.9: Our late fusion encoder-decoder architecture with modality-specific encoders for RGB and sparse depth. The final feature maps from both encoders are fed into a joint decoder where they are concatenated, then we apply convolution to mix features followed by transposed convolution to produce fused upsampled features at each resolution stage. Skip connections from both encoders are used to preserve high resolution details.

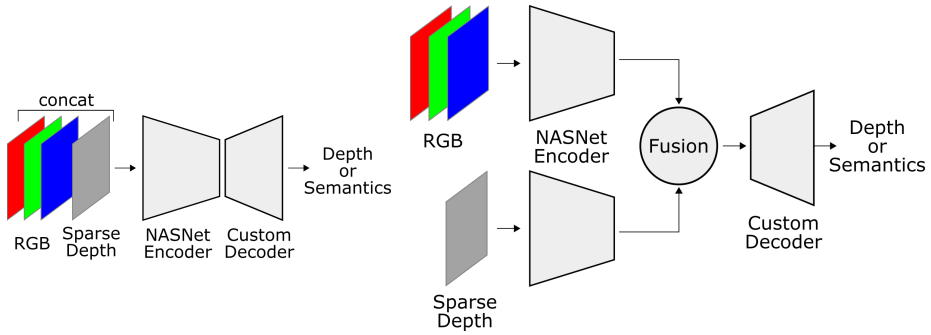
3.3 Method

Our goal is to efficiently fuse sparse depth and dense RGB data for the task of depth completion.

We first detail our encoder-decoder architecture and RGB+sD late-fusion strategy. Then, we introduce our sparse data learning scheme, where test time robustness against varying input sparsity can be achieved simply through training with varying sparsity, a form of data augmentation.

3.3.1 Network Architecture

Our network architecture is depicted in Figure 3.9. Each modality has its own encoder based on NASNet (Zoph et al., 2018). In the bottleneck at the lowest resolution, the feature maps of the two encoders are concatenated, then follows a convolution with ReLU activation to mix the features and a transposed convolution with ReLU activation for upsampling, resulting in fused upsampled features. For the next stages, the fused upsampled features are concatenated with the RGB and sparse depth features (skip connections like in U-Net(Ronneberger et al., 2015)) and again followed by convolution and transposed convolution until the final spatial resolution is reached. The last layer consists of a 1x1 convolution to regress the pixel-wise depth value. It is also followed by a ReLU activation to cap negative depth values. Note, that the last layer can be easily adapted to predict semantic segmentation by setting the output channel size to the number of classes.



(a) Early fusion (b) Late fusion (ours)

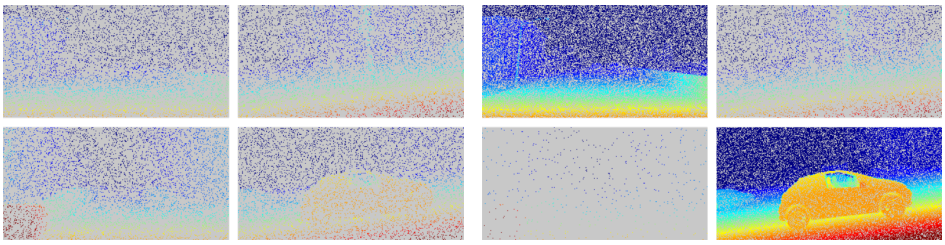
Figure 3.10: Visualization of early fusion (a) as opposed to our late fusion framework (b).

Encoder-Decoder. For dense data, state-of-the-art methods in semantic segmentation use encoder-decoder networks or dilated convolutions (Chen et al., 2018). While the latter significantly reduces the number of parameters, they are ill-conditioned for sparse data as dilated kernel with zeros between weights can miss available pixels in sparse data. In (Uhrig et al., 2017), a network without any downsampling is used with at most 11x11 kernels, leading to a rather small receptive field. As a consequence, Uhrig et al. (2017) perform similarly as classical local interpolation (Ku et al., 2018). As Zhang and Funkhouser (2018), we opt for an encoder-decoder with a larger receptive field, thus enabling the use of context for better data completion. Recent works since our publication have also adopted the encoder-decoder approach (Yang et al., 2019; Eldesokey et al., 2019; Xu et al., 2019).

The encoder part of our network is an adaptation of NASNet (Zoph et al., 2018) which is flexible and very efficient in terms of parameters vs. performance. We use the mobile version to fit real-time constraints and slightly modify it by removing batch normalization after the first strided convolution layer for the sparse depth branch. The latter is necessary because zero values of missing pixels falsify the mean computation of the batch norm layer. This was confirmed by Ma et al. (2019) after the publication of our work.

We build our own custom decoder with transposed convolutions for upsampling, normal convolutions, and copy and concatenate skip connections between the encoder and decoder stages of equivalent resolution like in U-Net (Ronneberger et al., 2015).

Late fusion of sparse depth and RGB. The scene is sensed with a camera and a depth sensor and we want our network to learn to use dense RGB and sparse depth information jointly for better prediction. A naive



(a) Fixed density of 0.1 (b) Varying density $]0, 1]$

Figure 3.11: Fixed vs. varying density in example training batches of 4. In (a) the density is fixed to 0.1, while in (b) the density is varied image-wise between 0 and 1, to increase test time robustness to different densities. In synthetic data, here on Synthia, we can artificially create different densities through uniform sampling.

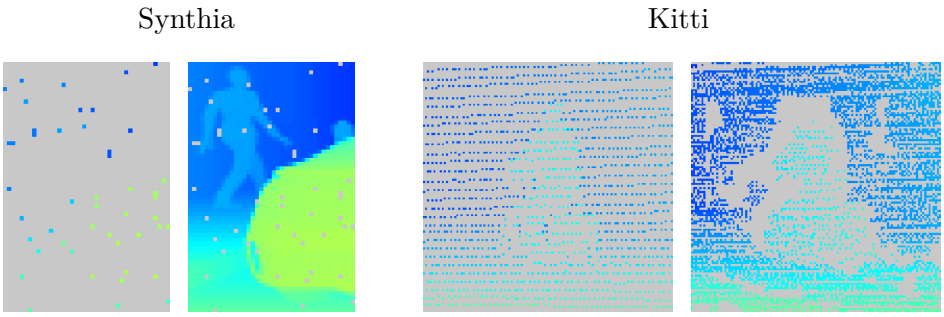
strategy consists of averaging separate predictions from each modality, but it is preferable to fuse features at an earlier stage to benefit from the complementarity of the modalities. One possibility, is to apply an early fusion like in (Ma and Karaman, 2018), where modalities are simply concatenated channel-wise and fed to the network (Figure 3.10a).

However, the modalities are different in nature (RGB intensities, distance values) and in order to reason from both at the same time, it appears preferable to transform them to a similar feature space before fusing them (known as late fusion, Figure 3.10b). A joint representation can be enforced by using element-wise addition of features coming from modality specific encoder networks (Chen et al., 2017b). However, we chose instead to use channel-wise concatenation with a following convolution to allow the two branches to provide information of distinct nature as in (Valada et al., 2017).

3.3.2 Sparse Data Training

Varying density. Existing research surprisingly only uses fixed density during training, although we found that training with varying density within a range of $]0, 1]$ naturally helps networks to be invariant to different densities, as experimented with synthetic data in Section 3.4.3.1. In our implementation, the density is spatially uniform, but differs from frame to frame in one batch (see Figure 3.11).

Another interesting proposal from DeVries and Taylor (2017) is to apply rectangles cut-out on the sparse depth input data. While this should force the network to use farther away features and thus greater context, it barely improved results in our experiment.



(a) observed $\mathcal{O}$ (b) unobserved $\mathcal{U}$ (c) observed $\mathcal{O}$ (d) unobserved $\mathcal{U}$

Figure 3.12: Examples of observed pixels $\mathcal{O}$ and unobserved pixels $\mathcal{U}$ for Synthia and Kitti. We call the available pixels in the sparse depth input the set of observed pixels $\mathcal{O}$ and define as unobserved pixels $\mathcal{U}$ the ones that are available in the ground truth, but not in the sparse input. While an inversion of the observed pixels gives us the unobserved pixels in the dense example of Synthia, the ground truth is sparse in Kitti. Thus, even though we predict a depth value for all pixels we can only compute the loss on the unobserved pixels $\mathcal{U}$ depicted in (d).

Losses. In depth completion, we observe only a subset $\mathcal{O}$ of all pixels in the sparse depth input. We predict the dense depth and then compute the loss on unobserved pixels $\mathcal{U}$, available in the ground truth, but not in the input. We show examples for Synthia and Kitti in Figure 3.12. Thus, we write the total loss $\mathcal{L}_{\text{total}}$ for a depth map as the mean pixel-wise error E_{pix} over unobserved pixels $\mathcal{U}$:

$$\mathcal{L}_{\text{total}} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} E_{\text{pix}}(u). \tag{3.1}$$

Other strategies, such as computing the loss on all pixels (unobserved and observed) or using a weighted sum of observed and unobserved pixels losses, worked less well. The interest of our choice is to favor learning to predict the depth at unknown pixels over learning to reproduce already measured data.

In accordance with Uhrig et al. (2017), we found the $\mathcal{L}_1$ loss to reach slightly better results than $\mathcal{L}_2$ for depth prediction. Like Ummenhofer et al. (2017), we train using inverse depth, measured in [1/km], because like this, we are able to represent infinity (e.g. the sky) as 0. We apply the $\mathcal{L}_1$ loss to the inverse depth output which is equal to the Kitti metric of inverse Mean Average Error (iMAE).

In order to obtain the values for depth d, we reverse the inverse depth d_{inv} where $d_{\text{inv}} > 0$ and set the output value to the maximum representable depth d_{max} where the network regresses to $d_{\text{inv}} = 0$ (non-activation). It

reads:

$$d = \begin{cases} d_{\text{inv}}^{-1}, & \text{for } d_{\text{inv}} > 0 \\ d_{\max}, & \text{for } d_{\text{inv}} = 0. \end{cases} \tag{3.2}$$

Following recent practices, we now think that a better choice would have been clip the maximum resulting value of depth d to 85m to stabilize training, inspired by newer related works (Van Gansbeke et al., 2019). Similarly, Ma et al. (2019) clip at a minimum of 0.9m. These values can be physically motivated, because LiDAR measurements become noisy above 85m and below 0.9m is too near to the sensor.

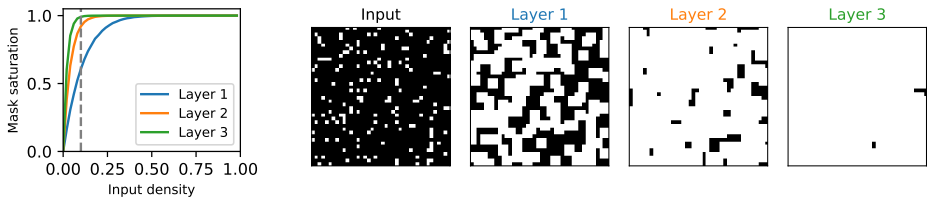
3.3.3 Analysis of Validity Mask

A validity mask (Uhrig et al., 2017; Luo et al., 2016) is a binary matrix of same size as the input data, with ones indicating available input data and zeros elsewhere. To propagate the mask through the network, Uhrig et al. (2017) use a max pooling of same kernel size k and stride s as the corresponding feature convolution (see Figure 3.5b). Intuitively, the resulting mask expresses whether the current input data seen by the filter contains at least one valid pixel. The authors further normalize convolution by the number of valid input pixels to rescale valid outputs accordingly. The drawback of such an approach is that the scaling can over-activate highly-upscaled features at lower density, e.g. when there is only one valid pixel.

In Figure 3.13a, we can see that the mask saturation (the percentage considered as valid) increases with input density as expected, but reaches almost full saturation after only a few layers. This means that the validity information is not useful in the later layers which is visible in Figure 3.13b, showing an example of such a validity mask and how it is transformed after a few layers. Another consequence is that the network tends to produce blurry outputs as seen in Figure 3.14. We interpret that this is a consequence of the normalization phase on the number of valid pixels, which processes a mask with only one valid pixel in the same way as a fully valid mask. In an attempt to address this issue, we tested average pooling to preserve the ratio of valid/invalid pixels in the filter. However, this did not improve results.

We tested to let the network learn how to use the validity mask by concatenating the actual validity mask channel-wise to the features before each convolution. While it improves performance on small networks, we observed no improvement with validity masks for large networks such as NASNet.

Tests confirmed our analysis that, without any validity mask, large networks still manage to learn effective features on sparse data. Consequently, we do not use any validity mask.



(a) Mask saturation

(b) Mask at 0.1 density

Figure 3.13: Saturation of the validity mask for different input densities with max pooling as Uhrig et al. (2017) (here 3 convs: stride 1, kernel size 3). (a) For density ≥ 0.3, the mask is saturated after first conv. (b) Even at only 0.1 density, the validity mask is mostly saturated from layer 2 and thus not informative.

3.4 Experiments

To evaluate our method we carried out experiments on two tasks: depth completion (Section 3.4.3) and semantic segmentation (Section 3.4.4). For both tasks we used either sparse depth (sD) LiDAR input, dense RGB input, or a fusion of both, and tested on both synthetic (Synthia) and real data sets (Kitti, Cityscapes).

We introduce the data sets in Section 3.4.1. On synthetic dataset Synthia, we benchmark the early vs. late fusion, described in Section 3.4.3.1. In Section 3.4.3.2, we first compare the performance of our approach to others on Kitti Depth Completion benchmark and then carry out a low-resolution LiDAR ablation study, where we reconstruct high quality depth maps from as few as 8 LiDAR layers. Finally in Section 3.4.4, we extend to semantic segmentation and prove that our method can segment in the camera image space based on sparse depth input only. Furthermore, we show that segmentation performance significantly improves when fusing RGB and sparse depth compared to the RGB only baseline.

3.4.1 Datasets

Synthia. Ros et al. (2016) use the Unity game engine and provide RGB, depth and semantics for urban and highway scenarios with pedestrian and cars. We use summer sequences 1, 2, 4 and 6 for training and sequence 5 for testing (all views). With our split the training/validation/testing sets contain 28484/1500/6296 frames, (bottom/center) cropped and rescaled to 320x160 pixels.

Sparse depth input is simulated through uniform sampling, by setting different ratios of pixels to zero, which results in different densities. A density

of 0.1 means that 10% of the pixels are available and 90% are not.

Kitti. The Kitti depth completion dataset provides RGB, raw LiDAR data (64 layers HDL-64E) projected into the image plane, and sparse ground truth from the accumulation of LiDAR point clouds with stereo consistency check (Uhrig et al., 2017). Comparison against other methods (Section 3.4.3.2) is performed at full-resolution (1216x352), but cropped (bottom/center) and rescaled to 608x160 to reduce training time elsewhere.

Depth being sparse, we use max pooling to downsample it to avoid loosing any points (common resize methods such as nearest interpolation would take zeros into account and corrupt the output).

Cityscapes. Cityscapes (Cordts et al., 2016) provides RGB and stereo disparity from German cities, with 20000 coarse and 3000 fine semantic annotations for training. We resized the data from 2048x1024 to 512x256 to reduce computation.

3.4.2 Implementation

We use the Tensorflow library for implementation and provide architecture details in Appendix A. As optimizer we use Adam with learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

All experiments have been carried out on a GTX 1080 Ti (11GB), except the experiments for the Kitti benchmark at full resolution which have been trained on a Tesla V100 (16GB) in order to fit into memory.

3.4.3 Depth completion

We first evaluate on synthetic data (Synthia) and then on real data (Kitti). The metrics are from the Kitti benchmark: Mean Average Error (MAE, $\mathcal{L}_1$ mean over all pixels), Root Mean Square Error (RMSE, $\mathcal{L}_2$ over all pixels), both in mm, as well as their inverse depth counterparts iMAE and iRMSE in 1/km between prediction $\hat{y}$ and ground truth y averaged over the number N of evaluated pixels. For the experiment on synthetic data in Table 3.1 we also report the δ-metric as defined by Eigen et al. (2014), which is the percentage of relative errors inside a certain threshold $\epsilon = \{1.05, 1.10, 1.25, 1.50\}$:

$$\delta = \frac{1}{N} \sum_i (\delta_i < \epsilon), \qquad \delta_i = \max\left(\frac{\hat{y}_i}{y_i}, \frac{y_i}{\hat{y}_i}\right). \tag{3.3}$$

3.4.3.1 Synthetic Data (Synthia)

Fixed Density. For this experiment we train and evaluate our method on a very low input pixel density of 0.02 (only 2 percent of pixels available). While the density is fixed during training, the valid pixel positions

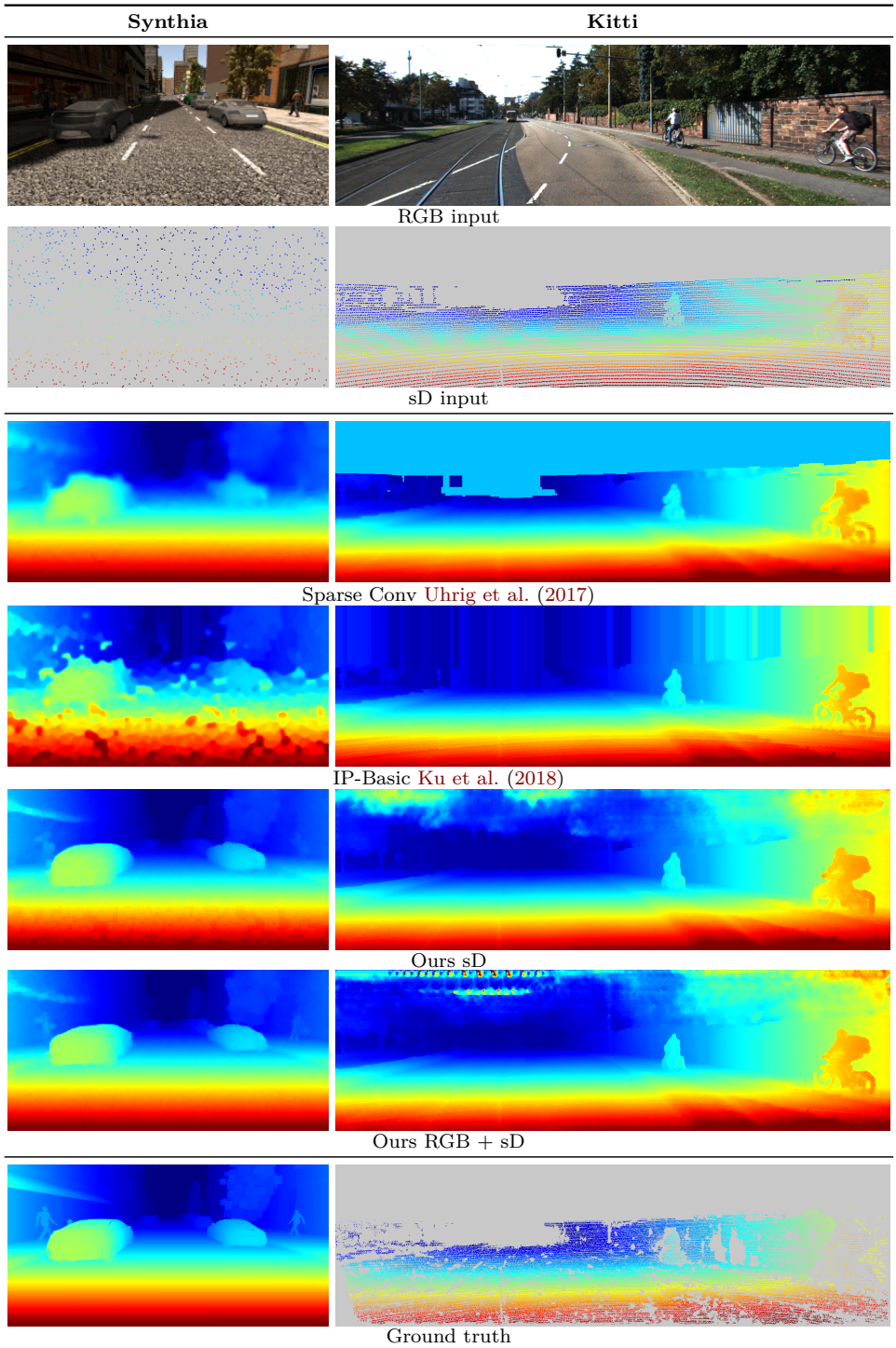


Figure 3.14: Qualitative results for depth completion on Synthia (synthetic) at 0.02 density and Kitti (real) validation set with 64 layers LiDAR.

Input	iMAE	$\delta 1.05$	$\delta 1.10$	$\delta 1.25$	$\delta 1.50$
RGB	13.56	0.56	0.69	0.85	0.92
sD	4.05	0.86	0.91	0.95	0.97
RGB + sD (Early fusion)	4.37	0.82	0.89	0.95	0.97
RGB + sD (Late fusion)	**2.96**	**0.87**	**0.92**	**0.96**	**0.98**

Table 3.1: Depth completion on Synthia at input depth density of 0.02. For iMAE lower is better, for δ-metric higher is better, indicated numbers are thresholds. While sparse depth is clearly the most important modality for depth prediction, late fusion can considerably improve the results.

are uniformely sampled. We train for 30 epochs and cherry-pick the best weights and report the iMAE as well as the δ-metric in Table 3.1. While sparse depth (sD) alone (4.05) performs better than RGB alone (13.56), proving that our network handles sparse data efficiently, the best results are obtained with the late fusion of RGB + sD (2.96). This is probably because the network can use the learned visual features from RGB to fill the holes in the sparse depth map, combining those complementary modalities efficiently. Late fusion clearly outperforms early fusion confirming that one network branch per modality is needed to map modalities to a similar feature space before fusion. Early fusion performs approximately as good as sparse depth only suggesting that the network simply ignores the less informative input modality. Qualitative results in Figure 3.14 exhibit sharp and precise completion for sD and even better for RGB+sD (notice the pedestrians). While Uhrig et al. (2017) perform decently, the result of Ku et al. (2018) is sharp but chaotic due to the very low density. Note that sparse depth is much more important as modality than RGB, even at this low sparsity level, because the input is a subset of the ground truth and depth prediction from RGB is a challenging task. This effect is even stronger in the case of the Kitti dataset, because the ground truth is constructed with imperfect LiDAR measurements instead of perfect synthetic data as in Synthia. Thus, giving LiDAR as input enables the network to overfit to errors in the LiDAR-based ground truth. Another consequence is that the network cannot be trained to perform depth completion outside of the field of view of the depth sensor, because ground truth is never available in this area (see sparse ground truth for Kitti in last row of Figure 3.14). This is why in Section 3.4.3.2 we conduct an ablation study: LiDAR layers are removed to test the robustness to lower density and the extrapolation capabilities of the network.

Varying Density. We further investigate the influence of different input densities at test time and plot results in Figure 3.15 for sparsity invariant CNNs (Uhrig et al., 2017) trained as in the paper with a fixed density (here of 0.1), our method trained at fixed density of 0.1, and our method trained at

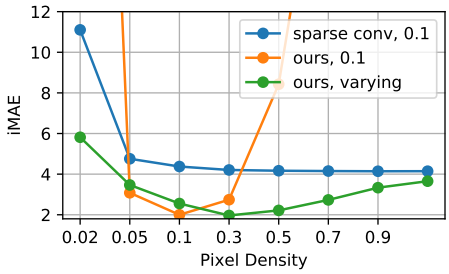


Figure 3.15: Test results on Synthia for depth completion with sparse depth only input at different densities. We compare sparse convolution (Uhrig et al., 2017) trained at fixed density of 0.1 (blue), our network trained at fixed density of 0.1 (orange) and our network trained at varying densities between 0 and 1.

varying density randomly chosen in $]0, 1]$ per image. Despite being trained at fixed density, the performance of Uhrig et al. (2017) is almost perfectly stable over different densities at test time, except for the lowest of 0.02. Our method trained at fixed density of 0.1 achieves much better test results close to the training density. However, the error increases drastically at lower or higher density. The network thus specializes in densities seen during training. However, when we train our network at varying density between 0 and 1, it gets very robust and we obtain better results than Uhrig et al. (2017) at all densities including at the lowest density of 0.02.

Results demonstrate that our method with varying training density could perform under a large variety of sensor densities which has great implications for LiDAR applications. For the other experiments, where train and test data have the same density, we use a fixed density to obtain the best results at test time.

3.4.3.2 Real Data (Kitti)

Depth Completion benchmark. In Table 3.2 we report the best published methods from the Kitti benchmark. At the time of submission, we ranked first. Note that previous methods from Uhrig et al. (2017) and Ku et al. (2018) do not use additional dense RGB input. We outperform them, even when not using RGB. Since our publication, many more methods have been published, as described in Section 3.2. All but one of these recent methods outperform us in terms of RMSE. However, we are still comparable to the state-of-the-art on the other metrics. Training directly with RMSE as loss function, as done by other works, would certainly help to optimize for this metric. While Eldesokey et al. (2019), Xu et al. (2019), Van Gansbeke et al. (2019) and Qiu et al. (2019) explicitly use confidence

Method	Input	iRMSE	iMAE	RMSE	MAE
Pre-submission					
Uhrig et al. (2017)	sD	4.94	1.78	1601.33	481.27
Ku et al. (2018)	sD	3.78	1.29	1288.46	302.60
Our submission					
Jaritz et al. (2018b)	sD	2.60	0.98	1035.29	248.32
Jaritz et al. (2018b)	RGB+sD	2.17	0.95	917.64	234.81
Post-submission					
Cheng et al. (2018)	RGB+sD	2.93	1.15	1019.64	279.46
Yang et al. (2019)	RGB+sD	2.10	0.85	832.94	203.96
Eldesokey et al. (2019)	RGB+sD	2.60	1.03	829.98	233.26
Ma et al. (2019)	RGB+sD	2.80	1.21	814.73	249.95
Xu et al. (2019)	RGB+sD	2.42	1.13	777.05	235.17
Van Gansbeke et al. (2019)	RGB+sD	2.19	0.93	772.87	215.02
Qiu et al. (2019)	RGB+sD	2.56	1.15	758.38	226.50
Chen et al. (2019b)	RGB+sD	2.34	1.14	752.88	221.19
Cheng et al. (2019)	RGB+sD	2.07	0.90	743.69	209.28
Tang et al. (2019)	RGB+sD	2.25	0.99	736.24	218.83

Table 3.2: Depth completion performance on the Kitti benchmark. For all metrics, lower is better. We split into methods before and after our submission to establish chronological order and highlight the best results at submission time in blue and the best post-submission results in red.

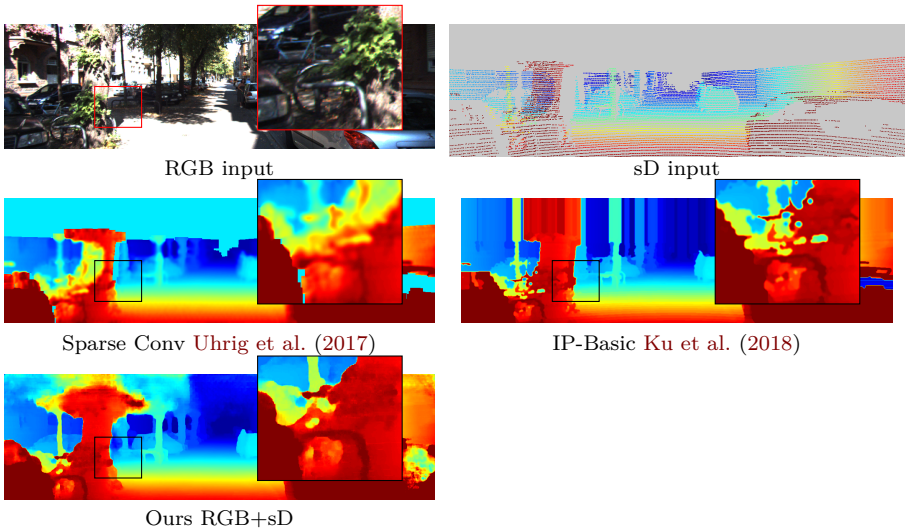


Figure 3.16: Qualitative results from public Kitti Depth Completion benchmark (recolored).

or the auxiliary task of surface normal prediction to improve performance, Chen et al. (2019b) and Tang et al. (2019) make pure network architecture improvements with 2D-3D fusion blocks and an encoder-decoder late-fusion approach similar to ours, but, among other changes, with a second decoder which purpose is to guide the dense depth decoder.

Figure 3.16 displays the visual output (recolored) from the benchmark website (test set). Classical morphological interpolation as IP Basic (Ku et al., 2018) is favored by the dense input of the 64 layers LiDAR. It produces very sharp output but fails logically to reconstruct shapes when the density is low. Sparse convolution (Uhrig et al., 2017) does better on shapes by integrating learned priors, but the output is rather blurry. These are expected counterparts as explained in Section 3.3.3.

Because the ground truth is sparse and only inside the LiDAR field of view, the network is never supervised regarding its prediction at the top of the image. This is better understood looking at results from the validation set (Figure 3.14) as we can display the ground truth along with the results. We use full resolution 1216x352, batch size 8 and train 20 epochs.

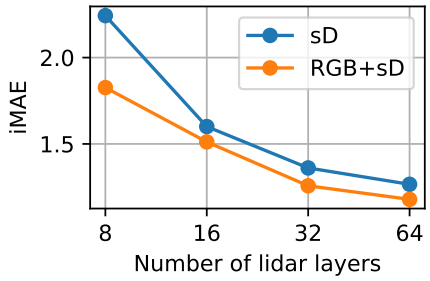
LiDAR Ablation Study. As the ground truth is obtained from LiDAR and because inferring the depth from monocular vision is an ill-posed task as stated by Eigen et al. (2014), the depth modality is much more important than RGB. To compensate for that and to give a clue which depth map precision can be obtained with fewer layer LiDARs, we subsample the 64 layers of Velodyne LiDAR data to simulate LiDARs with 8, 16 and 32 layers[1]. This study is valuable to explore possible applications for low-resolution, low-cost LiDARs such as the Valeo Scala series, produced for the mass market.

In Figure 3.17a we can see that by decreasing the number of input layers the dense depth map prediction deteriorates as expected. The RGB input always improves, especially at lower densities. Qualitative results in Figure 3.17b are remarkable, even with only 8 layers, i.e. 0.008 density. Note reconstruction of the bike in the foreground.

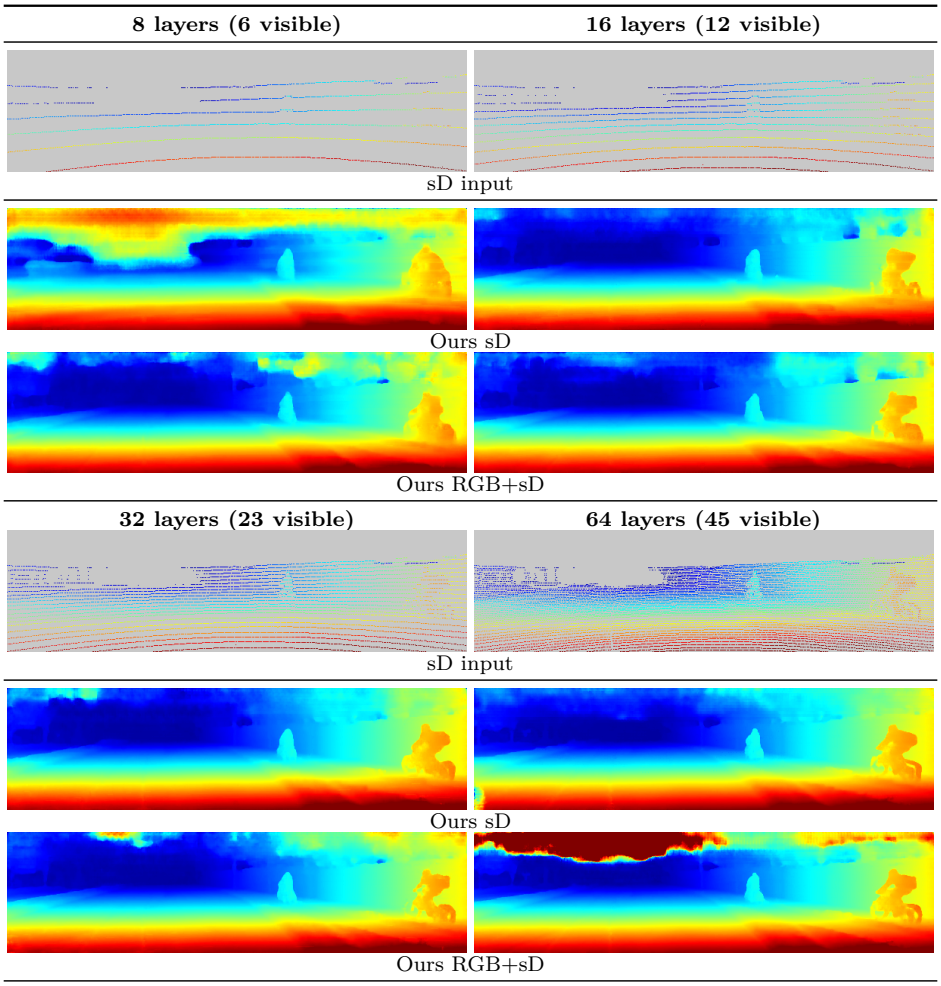
3.4.4 Semantic Segmentation

In this section we investigate how sparse depth input can improve semantic segmentation by evaluating our method on synthetic data (Synthia, 13 classes) and real data (Cityscapes, 19 classes). We do not use the Kitti segmentation benchmark, because the corresponding LiDAR point clouds are not provided and the number of frames for segmentation ground truth is very small (200).

[1]We subsampled every 2nd, 4th, and 8th layer to simulate 32, 16 and 8 layers, untwisted data linearly using the car speed from IMU, and projected into the camera image plane.



(a) iMAE error



(b) Qualitative results

Figure 3.17: Depth completion with simulated fewer layer LiDARs (down-sampling of 64 layers input). The graph in (a) shows that fusion (RGB+sD) is always better than depth only (sD). At only 8 layers, RGB+sD fusion improves by a large margin over depth only. Qualitatively, shown in (b), even with only 8 layers our method completes the depth map remarkably well.

Input	mIoU
RGB (baseline)	63.47
sD	57.10
RGB + sD (Early Fusion)	65.68
RGB + sD (Late Fusion)	**70.74**

(a) Synthia (synthetic)

Input	mIoU
RGB (baseline)	50.13
sD	44.18
RGB + sD (Early Fusion)	50.10
RGB + sD (Late Fusion)	**57.82**

(b) Cityscapes (real)

Table 3.3: Semantic segmentation on (a) Synthia (0.3 uniform sparse depth) and (b) Cityscapes (stereo depth) exhibit similar performance. RGB is the most important modality, but fusion with sparse depth always improves performance.

Note that our goal is not to improve state-of-the-art results in segmentation, but rather to compare against a RGB-only baseline. In the literature, semantic segmentation is usually carried out with RGB only and we prove that our method outperforms the baseline using additional sparse depth data as input.

To adapt our network to semantic segmentation, we modify the last layer by simply setting the number of output channels equal to the number of classes, instead of the 1-channel depth regression, and train with a cross entropy loss. Note, that we do not investigate the multi-task setting with semantic segmentation and depth regression at the same time, i.e. when we train our network on the semantic segmentation task, we do not let it predict dense depth. We report the mean Intersection over Union (mIoU), first computed per class and then averaged over classes.

3.4.4.1 Synthetic Data (Synthia)

For sparse depth, we use a pixel density of 0.3, close to the 64 Velodyne LiDAR (0.27 inside the FOV) and trained for 30 epochs with a batch size of 16. The results in Table 3.3a indicate, as expected, that texture and intensity from RGB data carry more semantic information than depth. As for depth completion, RGB+sD late fusion works best while early fusion fails to integrate depth features and reaches only slightly better output as RGB only. In Figure 3.18, sparse depth only shows very acceptable results but fails as expected on lane markings and far away buildings. With our late RGB+sD fusion the network better reconstructs the shape of cars.

3.4.4.2 Real Data (Cityscapes)

On real data (see Table 3.3b), we obtain similar results relatively to Synthia, although the nature of the depth data provided in Cityscapes is different: sparsity in patches, unscaled disparity from stereo camera instead of metric distance. We trained 50 epochs on coarse labels and then 50 epochs on fine

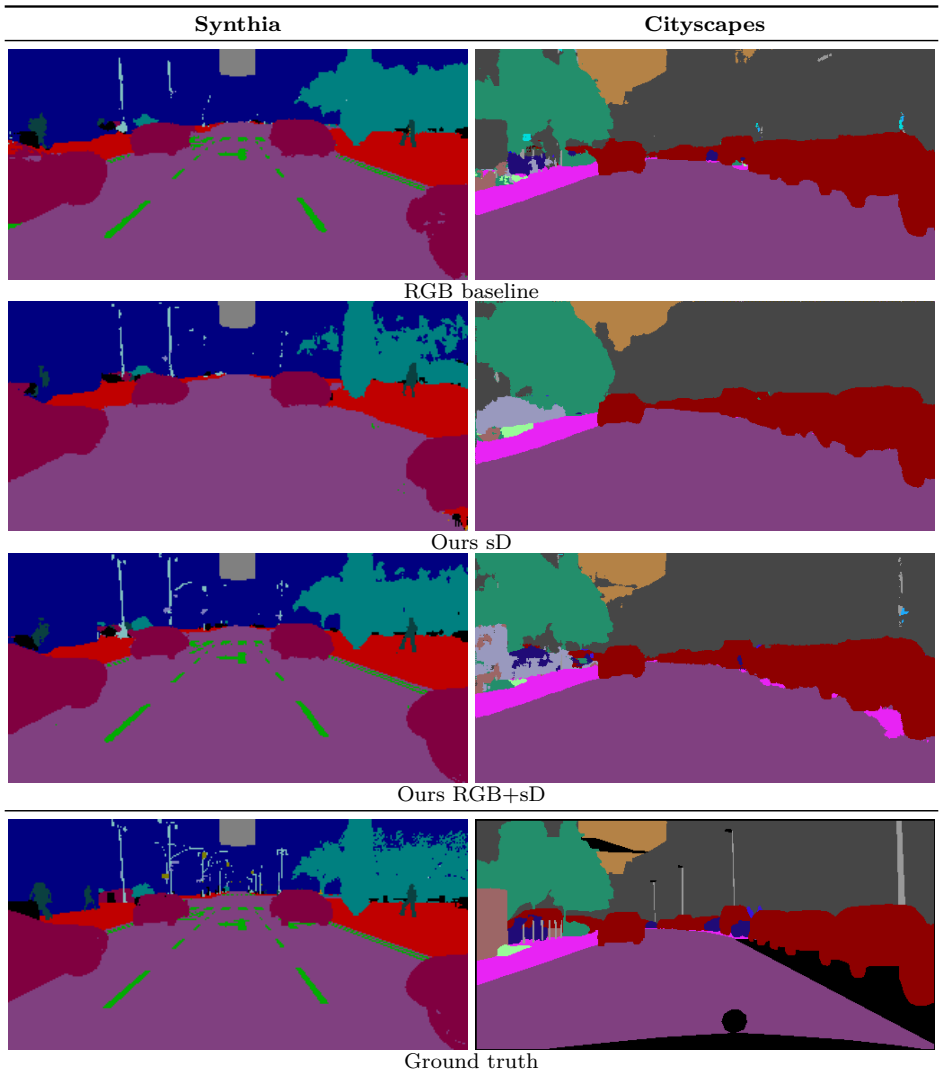


Figure 3.18: Qualitative results for semantic segmentation on the synthetic Synthia and the real Cityscapes dataset. In Synthia, we use a fixed sparse depth map density of 0.3. Note, that the network is, as expected, not able to predict lane markings using only sparse depth (sD) input. For Cityscapes, the input is of different nature and very dense (unscaled disparity from stereo camera). For both datasets, it is interesting to note that semantic segmentation from depth only (sD) has comparable performance as the RGB baseline and that fusion (RGB+sD) leads to the best results.

ground truth using a batch size of 16. The qualitative results in Figure 3.18 are satisfying, although thin structures like light poles are never segmented. Again RGB+sD fusion yields the best results.

Our findings demonstrate that sparse depth can directly be input into a network for semantic segmentation and possibly other tasks such as object detection without first generating the dense depth map like is commonly done in RGB-D networks as for example in (Gupta et al., 2014). Additionally, our method works with various densities and sparsity types (Figure 3.4) given that the latter does not change between train and test.

3.5 Discussion

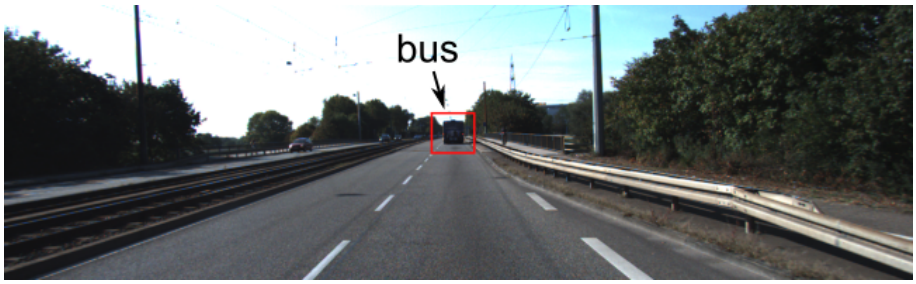
Review of our work. In retrospect, our design choices of using a U-Net style encoder-decoder network and fusing with RGB have been confirmed by the recent state-of-the-art (Yang et al., 2019; Eldesokey et al., 2019; Ma et al., 2019; Xu et al., 2019; Tang et al., 2019).

The late-fusion strategy has been used in multiple works (Yang et al., 2019; Eldesokey et al., 2019) and extended by the currently first ranking method (Tang et al., 2019). However, early fusion (Ma et al., 2019) and continuous fusion (Chen et al., 2019b) also achieve very good performance. In general, we can say that there is not one best fusion strategy; it depends on the network architecture.

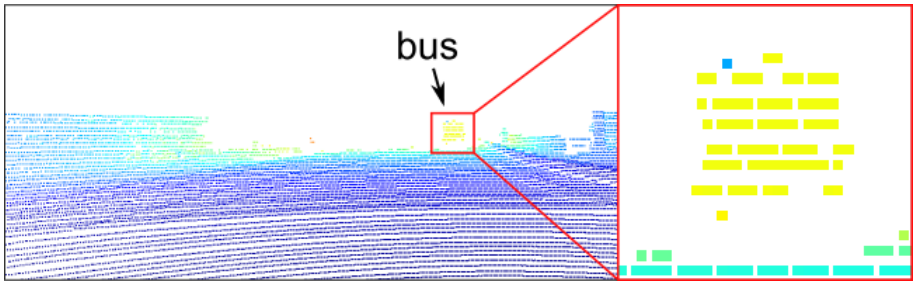
While the common choice for the encoder backbone is ResNet (He et al., 2016), we were the only ones using NASNet (Zoph et al., 2018), which we chose because of its better performance in classification benchmarks. However, today we think that ResNet is a good choice as it is much simpler to implement and the state-of-the-art has shown superior results with the ResNet backbone.

A limitation of our approach was that we sometimes encountered divergence of the loss. Other authors have apparently solved this problem by clamping the maximum predicted depth to 85m, as the LiDAR measurements become noisy above this threshold. We hypothesize that this bounds the loss and stabilizes training.

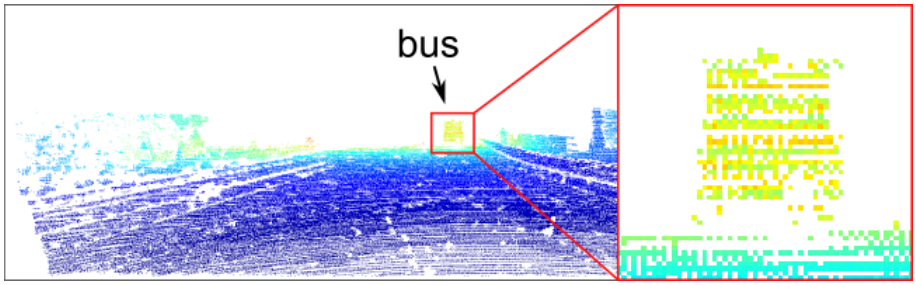
Data challenges. The ground truth data is generated automatically which can induce bias. In Kitti, the ground truth is obtained by accumulating 11 LiDAR frames: the current frame, the 5 preceding and the 5 succeding frames. In post-processing, the dense accumulated depth map is compared to the disparity map, obtained from stereo camera, and inconsistent pixels are removed from the ground truth depth map. This creates a bias: The current LiDAR frame is used as input data, but also in the ground truth. Thus, overfitting to the LiDAR can improve performance and on the contrary, learning the depth in a self-supervised fashion with left-to-right stereo



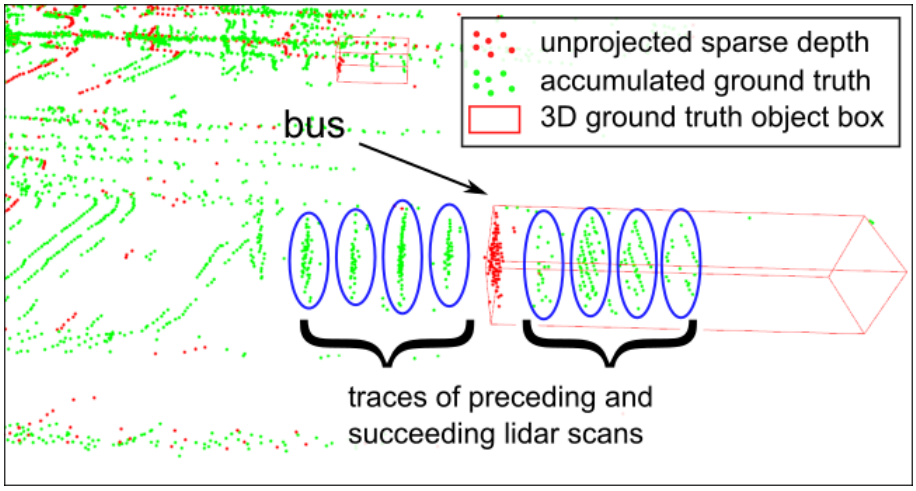
(a) Camera image



(b) Sparse depth from LiDAR (points are dilated for better visualization)



(c) "Dense" ground truth for learning depth completion



(d) Unprojection of depth maps to 3D, zoomed view from the right side of the bus.

Figure 3.19: Errors in dense ground truth due to accumulation. Although the 2D depth map on the bus seems consistent in (c), the errors become visible in (d) the unprojection to 3D. The locations of the bus at the preceding and succeeding time frames are still clearly visible, probably because the consistency check with stereo camera is limited to shorter distances up to ∼30m.

warping can not match the performance of supervised learning (Ma et al., 2019).

Apart from this bias, moving objects are problematic, because they lead to traces in the 3D data due to accumulation over several time steps. Although, when constructing the Kitti depth completion dataset, Uhrig et al. (2017) removed inconsistent pixels using stereo vision, there are limitations to that. One important case is distances beyond ∼30m, where the precision of stereo sharply decreases due to limited image resolution as was stated in (Wang et al., 2019a; You et al., 2020). We show such an example in Figure 3.19. One sees that if a moving object is far away, the 3D traces are still present in the ground truth which negatively influences training and induces a bias in evaluation. Solutions to that problem would be to remove moving objects from ground truth altogether. It would still be possible to learn depth, for instance from static cars (e.g. parked or waiting at red light). Another possibility is to use a second depth sensor for ground truth only. We could imagine wanting to densify a low cost, low resolution LiDAR, fusing with the dense camera image, where a high resolution LiDAR provides the depth completion ground truth.

Beyond depth completion. Depth prediction is an important low-level task, but usually not the final goal. Pseudo-LiDAR++ (You et al., 2020) focuses on 3D object detection: first, a depth map is predicted from stereo vision and refined using the point cloud from a 4-layer low-resolution LiDAR. Second, the refined depth map is unprojected to a 3D point cloud and fed into a 3D objection detection framework such as (Shi et al., 2019). The innovation lies in the connection of existing works in dense depth map prediction and 3D object detection from point clouds, as well in the analysis that if the final prediction space is 3D, the data should be lifted from 2D to 3D first. It is also very interesting from an industrial view point, because it possibly allows for a low-cost 3D perception system by combining camera and low-resolution low-cost LiDAR. We think that a possible disadvantage of (You et al., 2020) is that the 3D point cloud is an information bottleneck where visual appearance features from the image are lost. Lifting 2D image features to 3D can help to improve 3D final object detection as we show in the next chapter.

While *depth* completion is usually done in the 2D camera space in outdoor environments, *scene* completion of 3D point clouds uses indoor datasets, typically obtained from many RGBD frames. Therefore, one can either unproject a single depth map to 3D (Song et al., 2017) and then complete it in 3D, or construct a point cloud using many RGBD scans (Dai et al., 2018, 2019). Similar to depth completion, Dai et al. (2019) use incomplete ground truth, but use an even more incomplete input, obtained from fewer RGBD scans. We think that also for the outdoor setting it would be inter-

esting to extend from 2D depth completion to 3D scene completion, as the scene understanding tasks for applications such as autonomous driving are ultimately in 3D space.

3.6 Conclusion

We presented a network that fuses RGB and sparse depth with a late-fusion approach, using a U-Net inspired encoder-decoder network with NASNet encoder blocks. Following our study of sparse data, we do not use a validity mask, as we have found imperically that it does not help. On the opposite, removing batch norm led to improvements.

Our results on depth completion were better than those of the other published methods on the Kitti benchmark at the time of submission, but since have been outperformed by approaches that are often similar in architecture design, i.e. that use U-Net style encoder-decoder with fusion of RGB and sparse depth.

A study on low resolution LiDAR has shown that a qualitatively remarkable performance can be achieved with only 8 layer LiDAR which advocates for data-driven sensor fusion with RGB. Changing only the last layer, we also performed semantic segmentation and demonstrated important performance gains on synthetic and real datasets with our late fusion approach compared to RGB only.

With their 2D-3D fusion blocks, Chen et al. (2019b) drew attention to the problem of projecting 3D data into 2D space. Indeed, neighboring pixels in 2D are not necessarily close in 3D. Typically, pixels can be neighbors in 2D even though an occlusion boundary lies between them, which means that they are far apart in 3D. We address this problem in the next chapter, where we fuse data in 3D space by lifting 2D data to 3D, rather than projecting 3D data to 2D.

Compréhension 3D de scènes avec PointNet multi-vues

French Summary of the Chapter “Multi-view PointNet for 3D scene understanding”

Dans ce chapitre, nous fusionnons un nuage de point 3D et l’image 2D directement dans l’espace 3D pour éviter la perte d’informations dû à la projection. Pour cela, nous calculons les caractéristiques d’image issues de plusieurs vues avec un CNN 2D, puis nous les projetons dans un nuage de points 3D global pour les fusionner avec l’information 3D. Par la suite, ce nuage de point enrichi sert d’entrée à un réseau "point-based" dont la tâche est l’inférence de la sémantique 3D par point.

Chapter 4

Multi-view PointNet for 3D scene understanding

The contributions of this chapter were published in (Jaritz et al., 2019):

Jaritz, M., Gu, J., and Su, H. (2019). Multi-view PointNet for 3D scene understanding. In *ICCV Workshop 2019*.

The code is publicly available at:

https://github.com/maxjaritz/mvpnet.

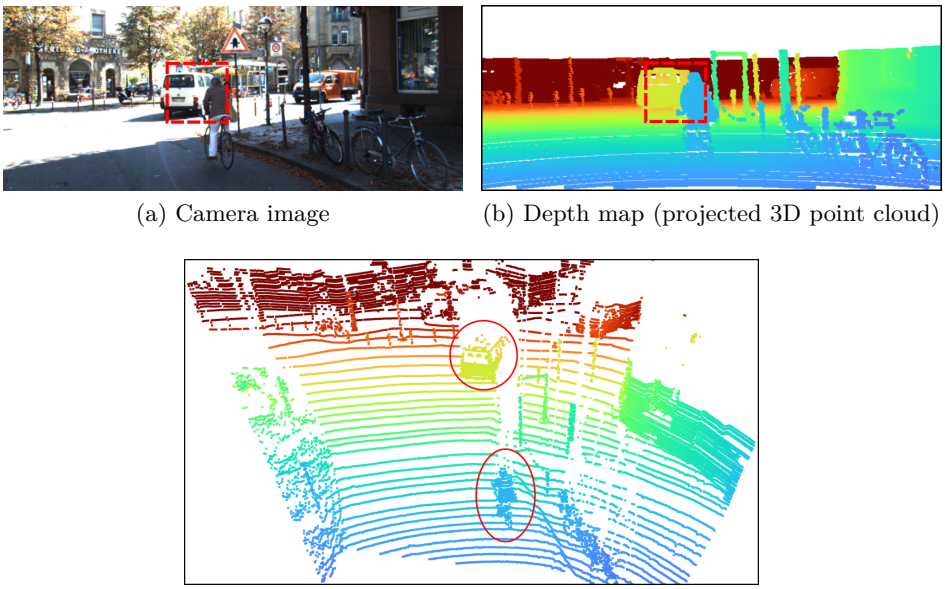
This work was done while visiting Hao Su's lab[1] at University of California, San Diego for 5 months. The lab's main research goal is to learn to perceive the 3D physical world and interact with it. Core areas of research are computer vision and graphics (large-scale dataset construction, 3D object/scene understanding), machine learning and generic AI (reinforcement and imitation learning) and robotics (object manipulation and realistic physical robot simulators).

The method presented in this chapter is applied to indoor instead of autonomous driving data with the motivation of exploring new 3D perception datasets.

[1] https://cseweb.ucsd.edu/~haosu/

Contents

4.1 Introduction **77**
4.1.1 Outdoor vs. indoor data 79
4.2 Related Work **81**
4.3 MVPNet **85**
4.3.1 Overview 85
4.3.2 View Selection 86
4.3.3 2D Encoder-Decoder Network 87
4.3.4 2D-3D Feature Lifting Module 87
4.3.5 3D Fusion Network 88
4.4 Experiments on ScanNet **89**
4.4.1 ScanNet Dataset 89
4.4.2 Implementation Details 89
4.4.3 3D Semantic Segmentation Benchmark 90
4.4.4 Robustness to Varying Point Cloud Density 93
4.5 Ablation Studies **93**
4.5.1 Number of Views 95
4.5.2 Feature Aggregation Module 95
4.5.3 Fusion 97
4.5.4 Stronger Backbone 97
4.6 Experiments on S3DIS **98**
4.6.1 S3DIS Dataset 98
4.6.2 3D Semantic Segmentation 98
4.7 Conclusion **99**



(a) Camera image (b) Depth map (projected 3D point cloud)

(c) 3D point cloud

Figure 4.1: Problems with regard to neighborhood definition in 3D-2D projection. In (a) we show the image and in (b) the projection of the 3D point cloud into the 2D image space, resulting in a (sparse) depth map. In (a) and (b), the van and the bike are neighbors, as highlighted by the red square. However, in (c), where the 3D point cloud is depicted, the objects are naturally separated. Data from Kitti (Geiger et al., 2013).

4.1 Introduction

In th previous chapter, we have proposed an architecture to fuse 2D images and 3D point clouds in 2D image space. The projection of the 3D point cloud into 2D image space represents a loss of information, because the neighborhood definition is richer in 3D than in 2D, which we show with an example in Figure 4.1. If the square in Figure 4.1a would be a convolutional kernel, then the local features of the van and the bike would be mixed which can lead to confusion and decrease the performance of semantic segmentation. This does not happen in 3D in Figure 4.1c, because the objects are well separated. In this chapter, our goal is to perform fusion of images and point clouds in 3D, rather than 2D space, to leverage the more exact neighborhood definition. This is especially helpful if the final task is three-dimensional, such as 3D object detection or 3D semantic segmentation, depicted in Figure 4.2. Those tasks are crucial to applications such as virtual reality and autonomous cars, because they need to operate in the 3D world.

The field of 3D perception is evolving at a fast pace driven by the introduction of new datasets – for example ScanNet (Dai et al., 2017), Stan-

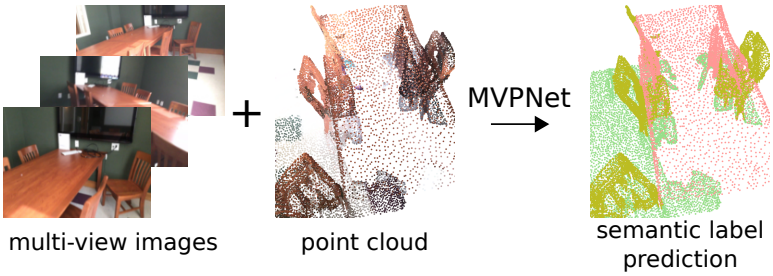


Figure 4.2: Our MVPNet (Multi-View PointNet) takes dense multi-view images and a sparse point cloud as input and fuses them to predict the semantic labels for each point.

ford Indoor 3D (Armeni et al., 2017), Semantic3D (Hackel et al., 2017), SemanticKITTI (Behley et al., 2019) – and powerful 3D networks – for instance PointNet++ (Qi et al., 2017b) or SparseConvNet (Graham et al., 2018).

However, efficiently fusing data from the 2D and 3D domains remains challenging, because there is no discrete one-to-one mapping between 2D and 3D data, and also the neighborhood definitions in 2D and 3D are different for convolution. More critically, while neighboring pixels are defined by the discrete grid, 3D points are defined at unstructured continuous locations. Additionally, there can be a discrepancy in resolution, usually occuring with 3D to 2D projection. For example, we have seen in Chapter 3 for the Kitti dataset, that when the point cloud from a Velodyne HDL-64 Lidar is projected into the camera image, it covers only 5.9% of the pixels.

In indoor datasets, 3D scenes are usually reconstructed from a sequence of frames (a video) that represents different views. While it is beneficial to incorporate detailed local information from partial views of the scene, it is challenging, because those views must be lifted into a canonical space where they can be fused in the global context and because the frames need to be selected so as to maximize information. Additionally, the number of views which we are able to process is limited by computational resources.

Point cloud based neural networks have been shown to generate powerful geometry cues for 3D scene understanding. However, some objects lack geometric saliency and cannot be distinguished by their shape, for example when they have flat surfaces such as doors, refrigerators and curtains. Therefore, additional color information should be leveraged, but recent results from Wu et al. (2019b) have shown that naively feeding colored point cloud (XYZRGB) to *point cloud based networks* does only marginally improve the performance over simple point cloud input (XYZ).

We argue that, because RGB cameras have much higher spatial resolution than 3D sensors in most realistic settings, it is better to compute image features in 2D first before lifting the 2D information to 3D. In this manner,

it is possible to gather *additional information* from higher resolution images and it is also *natural* from a sensor fusion perspective, to push modality centric features from different sources to 3D for their combination.

As different representations in 3D exist (voxel, point-cloud, multi-view, etc.), their respective scene understanding methods evolve in parallel. For voxel-based methods, there has been work on how to fuse geometry and image data coherently by unprojecting image features to 3D space (Dai and Nießner, 2018). However, for the point cloud domain, the common practice is to sparsely copy RGB information to points and there lacks a systematic exploration of how to conduct the fusion more effectively. In order to address this significant drawback of point cloud based methods, we propose MVPNet (Multi-View PointNet), where we first compute 2D image features on multiple, heuristically selected frames, then lift those features to 3D and adaptively aggregate them into the original point cloud (XYZ). Finally, the multi-view augmented point cloud is fed into PointNet++ (Qi et al., 2017b) for semantic segmentation.

The are four advantages of our method. First, the lifted 2D features contain contextual information thanks to the receptive field of the 2D network. Second, the complementary RGB and geometry features are jointly processed in natural, canonical 3D space. Third, thanks to lifting multiple views to 3D space, we are able to process multiple views at the same time in a local-to-global approach. And fourth, our framework is general as it is independent from the architecture of the 2D and 3D network backbone.

In summary, the key contributions are as follows:

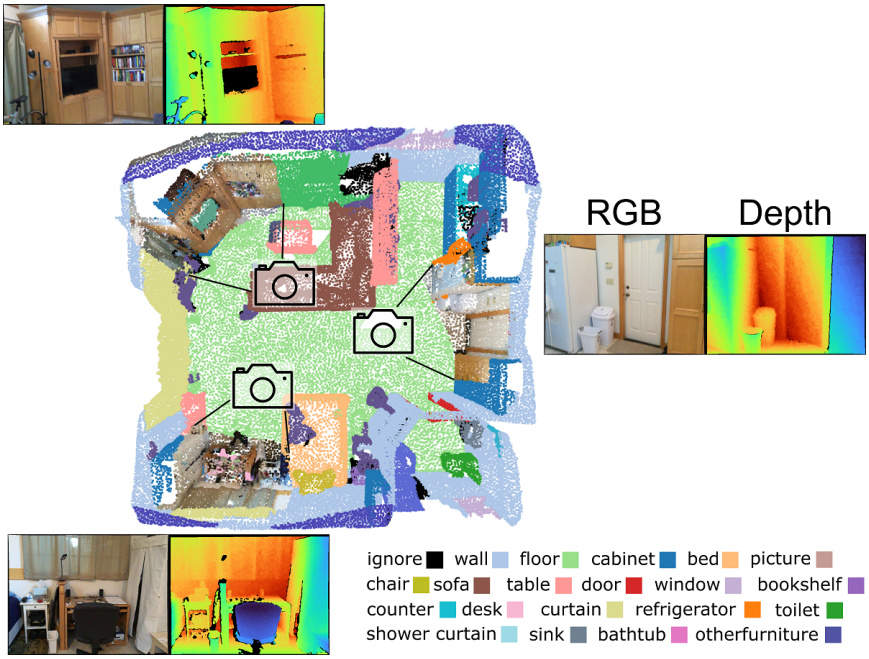
- We design an architecture that first computes 2D features for each view, then lifts those features to *canonical 3D point cloud space* and is followed by a 3D network to carry out the task of 3D semantic segmentation. This allows the fusion of complementary features from RGB and point cloud, and to jointly exploit multiple local views in a global context.

- We provide insights to the design choices in dense-2D/sparse-3D point cloud fusion based on extensive experiments, and showcase its excellent robustness to very sparse point clouds.

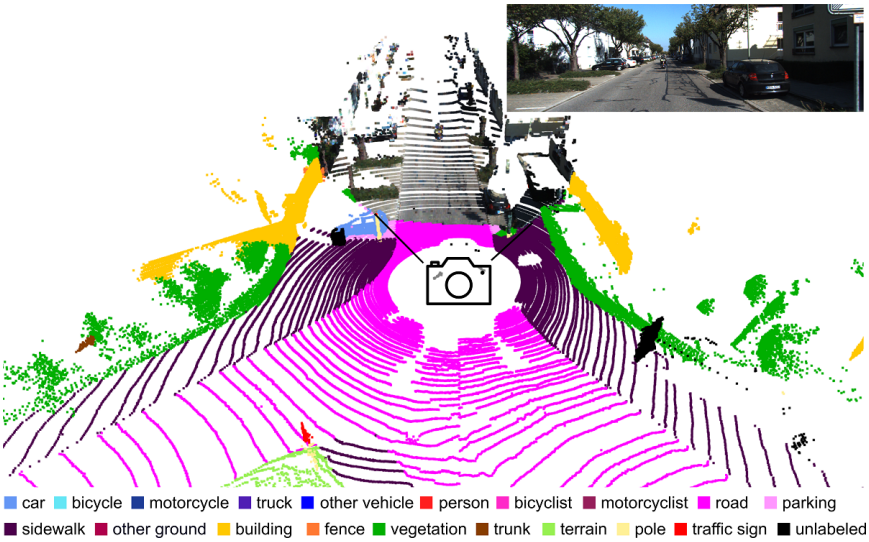
For the task of 3D semantic segmentation, our method outperforms all other 2D-3D fusion methods on the ScanNet dataset (Dai et al., 2017) and reaches competitive results on S3DIS (Armeni et al., 2017).

4.1.1 Outdoor vs. indoor data

In this chapter we focus on indoor datasets, as opposed to the rest of this thesis, where we use outdoor data from driving datasets. In Figure 4.3 and Table 4.1, we compare between the ScanNet indoor (Dai et al., 2017)



(a) ScanNet (Dai et al., 2017)



(b) SemanticKITTI (Behley et al., 2019)

Figure 4.3: Global point clouds, colored with semantic labels and obtained from (a) reconstruction using many RGB-D frames, (b) 360° LiDAR. The projected RGB frames are shown in image colors. While there are (a) multiple views in ScanNet (only 3 out of 5577 views are shown), (b) there is only a single view in SemanticKITTI.

Characteristic	ScanNet	SemanticKITTI
Space	indoor	outdoor
Moving objects	✗	✓
3D semantic segmentation	scene-wise	frame-wise
3D sensor	Depth camera (Structure)	LiDAR (Velodyne HDL-64E)
Max. distance (specs)	5m	120m
Field of View (FOV)	same as RGB camera	overlap with RGB camera
3D points in RGB camera FOV	∼300k	∼20k

Table 4.1: Comparison of 3D semantic segmentation datasets ScanNet (Dai et al., 2017) and SemanticKITTI (Behley et al., 2019).

and SemanticKITTI outdoor dataset (Behley et al., 2019). Both are popular benchmarks for 3D semantic segmentation. While the 3D geometry in ScanNet was collected with a depth camera, in SemanticKITTI the 3D data comes from LiDAR that has much lower resolution. As ScanNet features scenes of apartment rooms and offices that are spatially bounded and static, it is comparably simple to reconstruct a dense mesh from the whole scene. SemanticKITTI on the other hand, consists of spatially unbounded driving scenes where reconstruction is much more complex, because of changing location of moving objects across frames. Scene-complete point clouds sampled from the reconstructed 3D mesh in ScanNet contain many different viewing angles of the same object, while scans in the SemanticKITTI benchmark are evaluated frame-wise and thus just show objects from a single perspective. When designing a 2D-3D fusion algorithm, ScanNet is thus more complex, as one has to take into account multiple 2D views to cover the 3D point cloud. As there are many views (in Figure 4.3a we only show 3 out of 5577), relevant ones have to be selected so as to maximize the information in them.

Despite those important differences between indoor and outdoor data, we think that it is beneficial to exploit them both, because it allows gaining a deeper understanding of general problems such as fusion. Moreover, there can be knowledge transfer, i.e. an algorithm developed in the indoor setting can be adapted to the outdoor setting and vice versa.

4.2 Related Work

2D to 3D Lifting. Several works have shown that lifting 2D features to 3D leads to better performance than just lifting RGB values (Liang et al., 2018; Dai and Nießner, 2018; Su et al., 2018; Chiang et al., 2019). They do so by establishing 2D-3D pixel-to-point correspondences to lift low-level features to 3D, as opposed to Qi et al. (2018) where only high-level 2D object proposals are lifted to 3D frustums.

Liang et al. (2018) gather 2D image features at nearest neighbor locations defined by a LiDAR point cloud to build a dense bird view map.

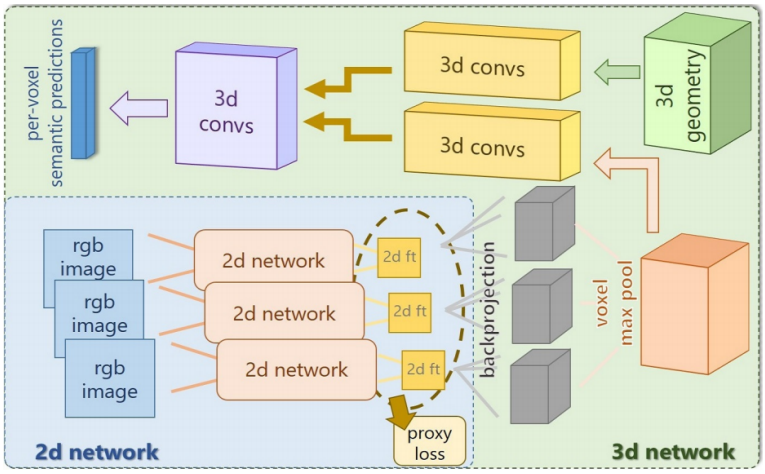


Figure 4.4: (Dai and Nießner, 2018) propose 3DMV where multiple views are selected and the corresponding RGB images fed into a 2D network. The 2D features are then individually unprojected (synonymous to backprojected) into a global 3D voxel representation of the scene. In order to combine the resulting 3D tensors, voxel max pooling is used. Then, a dense 3D CNN that takes RGB and geometry features as input produces the 3D semantic segmentation. Source: Dai and Nießner (2018).

3DMV (Dai and Nießner, 2018), as depicted in Figure 4.4, consists in unprojecting multiple 2D image feature maps to 3D voxel-volumes which are combined by max-pooling and then fed into a dense 3D CNN. One drawback of this approach is that it is limited in resolution and has a long runtime, because it uses dense voxels. SPLATNet (Su et al., 2018) takes point clouds and images as input and projects them on a permutohedral lattice for convolution and 2D-3D fusion. In our approach, we focus on fusing multi-view features with an aggregation module directly in the canonical point cloud space. As opposed to voxel (Dai and Nießner, 2018), permutohedral lattice (Su et al., 2018) and birdview (Liang et al., 2018) representation, we use point cloud space in this work. The advantage is that once all modalities are represented in a 3D point cloud, correspondence between two data points is precisely defined by distance in the continuous domain without discretization errors. Furthermore, operating directly on point clouds, a sparse representation, guarantees fast runtime and low memory consumption.

Concurrently with our work, Chiang et al. (2019) published a method similar to ours. While they share the basic idea with us – lifting 2D image features from local views to the global point cloud – their approach differs from ours in other aspects. In particular, Chiang et al. (2019) re-render multiple images with different viewing angles from the mesh which are fed into the quite large DeepLab network (Chen et al., 2018). On the contrary,

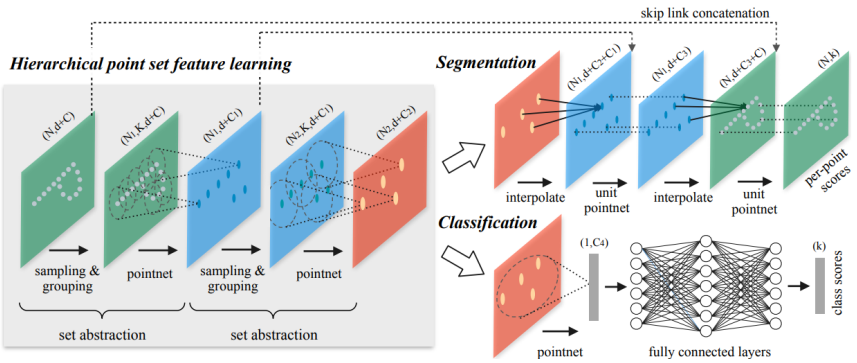
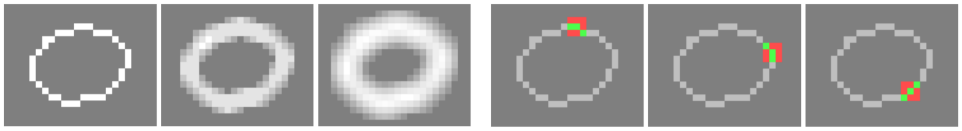


Figure 4.5: PointNet++ (Qi et al., 2017b) directly takes point clouds as input and uses downsampling, akin to CNNs, represented in the figure as the 'sampling & grouping' operation. For segmentation, the architecture follows an encoder-decoder approach with skip connections and interpolation for upsampling. Source: Qi et al. (2017b).



(a) Dense convolution (b) Sparse convolution (Graham et al., 2018)

Figure 4.6: Comparison of dense and sparse convolution. (a) Dense convolution. On the left is sparse input data, in the middle the result after applying standard 3x3 convolution with constant weights 1/9 and on the right the result after again applying that convolution. (b) Submanifold sparse convolution (Graham et al., 2018) is applied at three different locations. The red pixels are ignored and only the pixels shown in green contribute to the result. Source: Graham et al. (2018).

we use the original images with higher quality and feed it into a smaller, more efficient architecture that we derived from U-Net (Ronneberger et al., 2015). While Chen et al. (2018) use the mesh to propagate the unprojected 2D features into the 3D point cloud, we use a module that adaptively learns to aggregate features and handles overlapping views. Like us, Chen et al. (2018) use PointNet++ (Qi et al., 2017b) as 3D network.

3D Semantic Segmentation. The aim of 3D semantic segmentation is to predict a class label for every point in a 3D point cloud. While images are dense tensors, 3D point clouds are sparse and can be represented in multiple ways which leads to competing network families that evolve in parallel.

In *voxel-based networks* the raw point cloud data is transformed into a

discrete grid of cells. In practice, most of the cells are empty and only voxels that lie on the object surface are occupied. Given this sparsity, applying dense 3D convolution (Çiçek et al., 2016; Kamnitsas et al., 2017; Dai and Nießner, 2018) is not efficient. To speed up computation, reduce memory needs and in consequence allow for higher resolution grids, efficient sparse voxel representation can be leveraged. Therefore Riegler et al. (2017) use octrees and Graham et al. (2018) hash tables. The latter, which we denote as SparseConvNet in the following, defines a very efficient way to deal with sparsely populated voxels by restricting computations to active voxels. In Figure 4.6, we show a comparison of dense (standard) and sparse convolution. Dense convolution in Figure 4.6a dilates sparse data, similar to the dilation of the validity mask for depth completion in Figure 3.13. On the contrary, submanifold sparse convolution (Graham et al., 2018), shown in Figure 4.6b, only produces an output prediction on the original valid locations which, in the segmentation case, permits to restrict the prediction of labels to active locations.

Point cloud based networks directly consume point clouds. PointNet (Qi et al., 2017a) leverages shared Multi Layer Perceptrons (MLPs) to compute point-wise features and uses max-pooling to obtain features for the global point cloud. This works very well for part segmentation of single objects in the ShapeNet dataset (Chang et al., 2015). For whole scene analysis, PointNet++ (Qi et al., 2017b), shown in Figure 4.5, is more suited, because it has set abstraction layers using local neighborhood aggregation to create a hierarchical network structure akin to CNNs which scales much better to larger point sets. Wang et al. (2018), Li et al. (2018), Hermosilla et al. (2018) and Wu et al. (2019b) redefine convolution in continuous space, while still using MLPs as function approximator. On the other hand, convolution kernels can be defined with polynomials (Xu et al., 2018), linear functions (Groh et al., 2018), or by points (Thomas et al., 2019).

Lastly, *graph-based networks* convolve on the edges of a point cloud (Wang et al., 2019b; Verma et al., 2018). Although similar to convolution on the points, the intuition is here that the networks learn the relationships between points instead of associating features to the points themselves.

Let us compare SparseConvNet (Graham et al., 2018) and PointNet++ (Qi et al., 2017a). As SparseConvNet is highly efficient, it allows for very high resolution with typically only one point per voxel. Hence, discretization errors are small and computation is only carried out on the sparse voxels that are roughly equivalent to the points. Similarly in PointNet++, MLPs are applied directly (and only) on points. However, while SparseConvNet (Graham et al., 2018) has the advantage of a spatial kernel, e.g. of size 3x3x3, that is applied to the current voxel and its neighbors inside the kernel dimensions, PointNet (Qi et al., 2017a) first applies point-wise MLP, and only aggregates information across points in the following max-pooling.

In this chapter, we use *point cloud based networks* because of their in-

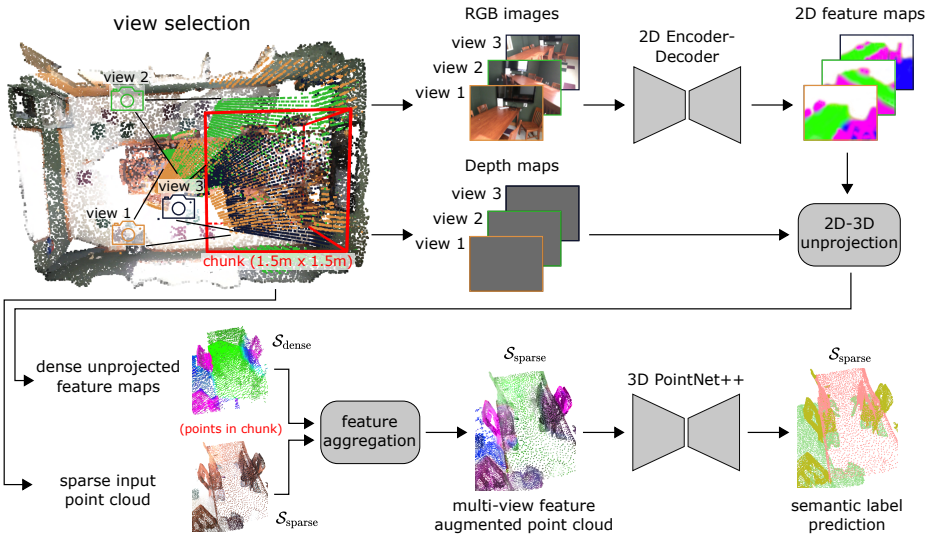


Figure 4.7: MVPNet pipeline overview. First, a fixed number of 2D views are selected so that the current region of interest (chunk) in the 3D scene is maximally covered. Then, the respective RGB images are fed into a 2D encoder-decoder to obtain feature maps of same size as the input images. The 2D feature maps are unprojected to 3D using the corresponding depth maps to form a dense point cloud. Then, the dense unprojected feature maps are aggregated into the sparse input point cloud to augment each point with 2D image features from nearby points. This feature augmented 3D point cloud is input into PointNet++ which predicts the final semantic labels.

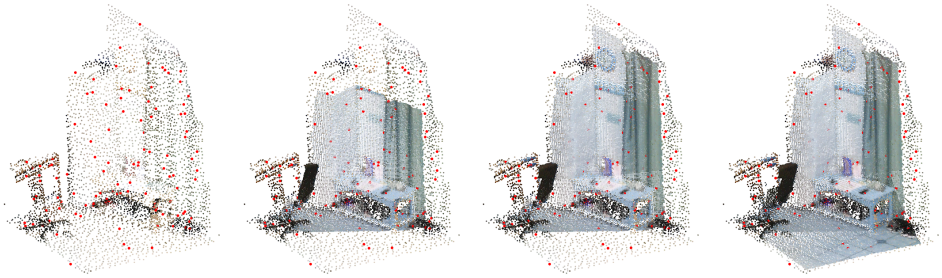
herent sparsity. Precisely, we use PointNet++ (Qi et al., 2017b).

4.3 MVPNet

Our MVPNet is designed to effectively fuse complementary information from multiple RGB-D frames and 3D point cloud in order to achieve better 3D scene understanding on real-world data, like ScanNet (Dai et al., 2017). The primary task is 3D semantic segmentation, where the goal is to predict a semantic label for each point in the input point cloud.

4.3.1 Overview

An overview of our architecture is shown in Figure 4.7. The data of each scene consists of a sequence of RGB-D frames and a point cloud. The input point cloud, denoted as $\mathcal{S}_{\text{sparse}}$, is sparse compared with the resolution of images. This can be seen in Figure 4.8 by comparing the density of the sparse point cloud with the unprojected views. Following Qi et al. (2017b), we divide the whole scene into chunks (around 90 chunks for an average



(a) sparse PC (b) 1st view (c) 2nd view (d) 3rd view

Figure 4.8: View selection: (a) visualizes the sparse input point cloud $\mathcal{S}_{\text{sparse}}$ of a chunk and its even coarser version (red points) used to compute the overlap with the RGB-D frames. (b), (c) and (d) show the 1st, 2nd and 3rd greedily selected view, respectively. Note that we call the point cloud from all unprojected RGB-D frames $\mathcal{S}_{\text{dense}}$, shown in (d), because it is much denser than the the original input point cloud $\mathcal{S}_{\text{sparse}}$, shown in (a).

scene). For each chunk, the M most informative views (RGB-D frames) are selected to maximize the coverage of the input point cloud (Section 4.3.2). Those views (RGB) are then fed into a 2D encoder-decoder network in order to compute M feature maps (Section 4.3.3). To augment the sparse input point cloud $\mathcal{S}_{\text{sparse}}$, pixels with valid depth in each 2D feature map are first unprojected to a 3D point cloud and then a dense point cloud $\mathcal{S}_{\text{dense}}$ is obtained by concatenating all the M unprojected point clouds. Given the image features associated with $\mathcal{S}_{\text{dense}}$, our feature aggregation module samples the k nearest neighboring points in $\mathcal{S}_{\text{dense}}$ and adaptively combines them to form the new feature for the point in $\mathcal{S}_{\text{sparse}}$ (Section 4.3.4). Finally, we feed the *multi-view feature augmented point cloud* to PointNet++ to process it from a 3D geometric perspective.

4.3.2 View Selection

In ScanNet (Dai et al., 2017), the RGB-D frames come as video stream with strong overlap between consecutive frames. It would be redundant and computationally expensive to process them all. Therefore, we make a selection of 1 to 5 views, which maximize contained information, to fuse with the point cloud of the scene.

In the preprocessing step, the overlaps between the scene point cloud and all the unprojected RGB-D frames of the video stream are computed. To reduce computation, we uniformly downsample the point cloud (red points in Figure 4.8). During training we use the overlap information to select the RGB-D frames on-the-fly with a greedy algorithm. The image which overlaps with the most yet uncovered points is selected. We found that this

straightforward but efficient method can achieve very high coverage even with few frames.

4.3.3 2D Encoder-Decoder Network

We feed the selected RGB images into a 2D encoder-decoder network based on U-Net (Ronneberger et al., 2015) to compute image feature maps. Note that it would be possible to add the depth channel as additional input. However, for effective RGB-D fusion a more sophisticated architecture such as the one for depth completion presented in Chapter 3 would have to be applied. For simplicity and faster run-time we decide to only use RGB in this chapter.

In our implementation, the size of the input image is equal to that of the output feature map, and fixed to 160×120. With this relatively low resolution, we found U-Net to be better suited in terms of memory, speed, and performance than other 2D semantic segmentation architectures such as DeepLabv3 (Chen et al., 2017a) and PSPNet (Zhao et al., 2017), optimized for a much higher resolution. We pretrain the 2D encoder-decoder network on the task of 2D segmentation on ScanNet in order to bootstrap the training of the whole pipeline. More details can be found in Section 4.4.2.

4.3.4 2D-3D Feature Lifting Module

In order to obtain the 3D coordinates for the feature maps that have been computed with the RGB images and the 2D encoder-decoder network, we unproject the corresponding depth maps using the camera instrinsics and poses. Consider M 2D feature maps of size $H \times W \times C_{\text{feat}}$, then each one is lifted to a point cloud of size $N_{\text{RGB}} \times C_{\text{feat}}$, where $N_{\text{RGB}} \leq HW$ is a hyperparameter that corresponds to the number of unprojected pixels in each RGB image. By concatenating all the M unprojected points together, we yield a dense point cloud $\mathcal{S}_{\text{dense}}$ of size $MN_{\text{RGB}} \times C_{\text{feat}}$.

For semantic segmentation, the labels have to be predicted for the input point cloud $\mathcal{S}_{\text{sparse}}$. Thus, we have to transfer the features from the unprojected point cloud $\mathcal{S}_{\text{dense}}$ to $\mathcal{S}_{\text{sparse}}$. Therefore, we use our feature aggregation module which includes a shared MLP inspired by Liang et al. (2018) in order to distill a new feature $\mathbf{h}_i$ for each point in $\mathcal{S}_{\text{sparse}}$ from its k nearest neighbors in $\mathcal{S}_{\text{dense}}$

$$\mathbf{h}_i = \sum_{j \in \mathcal{N}_k(i)} \text{MLP}(\text{concat}\left[\mathbf{f}_j, \mathbf{f}_{\text{dist}}\left(\mathbf{x}_i, \mathbf{x}_j\right)\right]) \tag{4.1}$$

where $\mathbf{h}_i$ is the distilled feature at point $\mathbf{x}_i$ in $\mathcal{S}_{\text{sparse}}$, $\mathbf{f}_j$ the semantic feature at one of the k nearest neighbors points $\mathbf{x}_j$ in $\mathcal{S}_{\text{dense}}$ defined by the neighborhood $\mathcal{N}_k(i)$, and $\mathbf{f}_{\text{dist}}(\mathbf{x}_i, \mathbf{x}_j)$ the distance feature between the two points

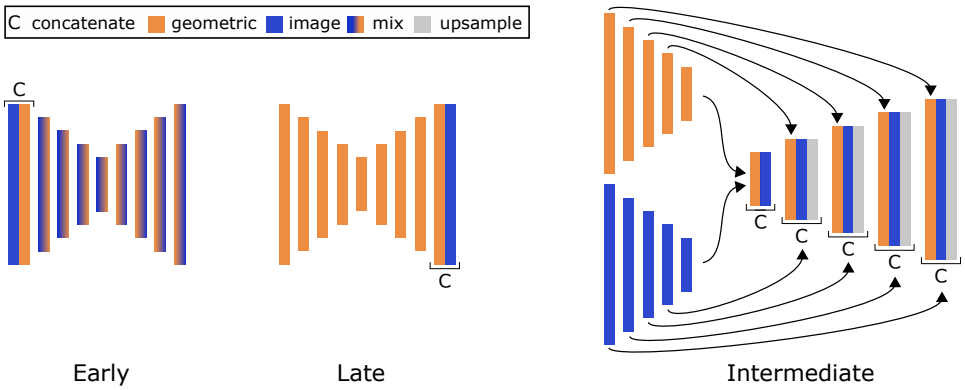


Figure 4.9: Proposed architectures to fuse geometric features and image features with PointNet++. We adopt early fusion, where the geometry (XYZ) and image features are concatenated at the input layer. The network fuses them which leads to mixed features for the remaining network. In the late fusion architecture, geometric and image features are concatenated in the last layer of PointNet++. In intermediate fusion, there are two PointNet++ expert encoders for geometric and image features, but a common decoder, similar to the architecture in Chapter 3.

which we define as

$$\mathbf{f}_{\text{dist}}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \text{concat}\left[\mathbf{x}_i - \mathbf{x}_j, \|\mathbf{x}_i - \mathbf{x}_j\|^2\right]. \tag{4.2}$$

We name *multi-view feature augmented point cloud* as the resulting point cloud augmented with the described features. Note that the whole 2D-3D feature lifting module is differentiable, which enables end-to-end training of our MVPNet.

4.3.5 3D Fusion Network

To fuse multi-view image features and geometry information, we employ PointNet++ (Qi et al., 2017b) as backbone. The original PointNet++ consumes both the coordinates and its corresponding features, such as normal or color. For 3D semantic segmentation, it encodes the input point cloud with set abstraction layers hierarchically, and decodes the output semantic prediction through feature propagation layers. The 3D coordinates of input points are concatenated to the output features of each set abstraction layer.

We adopt *early fusion*, depicted in Figure 4.9, where the image features are concatenated to the geometry (XYZ) and then given as input to PointNet++. Thus, the network is able to fully exploit the image features from a geometric perspective. We also investigated *intermediate fusion* and *late fusion*. In *late fusion*, the image features are concatenated after the final feature propagation layer in PointNet++, right before the segmentation head.

In *intermediate fusion*, we introduce separate encoder branches for geometry and image features whose outputs are then concatenated and fed into the decoder. Additionally, the decoder leverages the intermediate outputs of two encoder branches through skip connections.

4.4 Experiments on ScanNet

In this section we cover experiments on the ScanNet (Dai et al., 2017) dataset.

4.4.1 ScanNet Dataset

The ScanNet dataset (Dai et al., 2017), depicted in Figure 4.3a, features indoor scenes like offices and living rooms for which a total of 2.5M frames were captured with the internal camera of an iPad and an additionally mounted structure depth sensor (specifications detailed in Table 4.1). The data for each scan consist of an RGB-D sequence with associated poses, a whole scene mesh, as well as semantic and instance labels. There are 1201 training and 312 validation scans that were taken in 706 different scenes; most scenes were captured more than once. The test set contains 100 scans with hidden ground truth, used for the benchmark.

4.4.2 Implementation Details

For the task of 3D semantic segmentation, we follow the same chunk-wise pipeline as PointNet++ (Qi et al., 2017b). During training, one chunk ($1.5m \times 1.5m$ in xy-plane, parallel to ground surface) is randomly selected from the whole scene if it contains more than 30% annotated points. Random rotation along the up-axis is applied for data augmentation. During testing, the network predicts all the chunks with a stride of $0.5m$ in a sliding-window fashion through the xy-plane. A majority vote is conducted for the points that have predictions from multiple chunks.

We downsample the images and depth maps to a resolution of 160×120. Random horizontal flip is applied to augment images during training. We fix the number of unprojected points per RGB-D frame to 8,192. This means that only 43.6% of a total of 19,200 pixels (for a resolution of 160×120) are lifted to 3D. Note that the unlifted pixels are still essential for the 2D feature computation as they lie in the receptive field of lifted pixels in the 2D CNN that computes the features. Indeed, we will see in Section 4.5.3 that using only sparse RGB information, i.e. directly lifting color to 3D points which yields in a colored point cloud (XYZRGB), performs much worse than MVPNet which takes all dense RGB pixels into account through feature lifting.

Method	mIoU	bath	bed	shf	cab	chair	cntr	curt	desk	door	floor	other	pic	fridg	show	sink	sofa	table	toilet	wall	win
Su et al. (2018)	39.3	47.2	51.1	60.6	31.1	65.6	24.5	40.5	32.8	19.7	92.7	22.7	0.0	0.1	24.9	27.1	51.0	38.3	59.3	69.9	26.7
Dai and Nießner (2018)	48.4	48.4	53.8	64.3	42.4	60.6	31.0	57.4	43.3	37.8	79.6	30.1	21.4	53.7	20.8	47.2	50.7	41.3	69.3	60.2	53.9
Chiang et al. (2019)[1]	63.4	61.4	**77.8**	66.7	**63.3**	**82.5**	**42.0**	**80.4**	46.7	**56.1**	**95.1**	**49.4**	**29.1**	56.6	**45.8**	57.9	**76.4**	55.9	83.8	**81.4**	59.8
Ours	**64.1**	**83.1**	71.5	**67.1**	59.0	78.1	39.4	67.9	**64.2**	55.3	93.7	46.2	25.6	**64.9**	40.6	**62.6**	69.1	**66.6**	**87.7**	79.2	**60.8**

[1] Published posterior to submission of our contributions.

Table 4.2: Comparison of methods which use 2D-3D fusion on ScanNet 3D semantic label benchmark (hidden test set).

The backbone of the 2D encoder network is an ImageNet-pretrained VGG16 (Simonyan and Zisserman, 2015) with batch normalization and dropout. For ablation studies and submissions, we also experiment with VGG19 (Simonyan and Zisserman, 2015) and ResNet34 (He et al., 2016). The custom 2D decoder network is a lightweight variant of U-Net (Ronneberger et al., 2015). Batch normalization and ReLU are added after each convolution layer in the decoder.

For each chunk, 8,192 points are sampled from the input point cloud and augmented by the views selected by the method described in Section 4.3.2. For the feature aggregation module, we use a two-layer MLP with 128 and 64 channels. To predict the semantic labels for the *multi-view feature augmented point cloud*, we use PointNet++ with single-scale grouping (SSG) as our 3D backbone. However, note that our MVPNet strategy can be used with any point-based 3D network.

Each epoch consists of 20,000 randomly sampled chunks, and the batch size of chunks is 6. The network is trained with the SGD optimizer for 100 epochs. We use a weight decay of 0.0001 and a momentum of 0.9. The learning rate is 0.01 for the first 60 epochs, and then divided by 10 every 20 epochs. MVPNet is trained on a single GTX 1080Ti.

4.4.3 3D Semantic Segmentation Benchmark

The evaluation metric in ScanNet is the mean IoU (mIoU) over 20 classes. For submission to the benchmark, we ensemble 4 models of MVPNet, which consumes 5 views and uses ResNet34 as 2D backbone.

Comparison to 2D-3D Fusion Methods. In Table 4.2 we compare to methods that also fuse with 2D image features by showing performance on the hidden test set of ScanNet with per-class IoU. Our MVPNet outperforms all other methods that fuse with 2D data. Precisely, we achieve better performance (+24.8 mIoU) than SPLATNet (Su et al., 2018), which projects to a permutohedral lattice for convolution, and similarly (+15.7 mIoU) 3DMV (Dai and Nießner, 2018), which is limited in resolution and model size due to the memory-hungry dense 3D voxel representation.

We obtain slightly better results (+0.7 mIoU) than Chiang et al. (2019), who have a similar approach to us, but use re-rendered images from the

	Method	Fuses 2D	Representation	mIoU
Qi et al. (2017b)	PointNet++	✗	Point cloud	33.9
Su et al. (2018)	SPLATNet	✓	Point cloud	39.3
Dai and Nießner (2018)	3DMV	✓	Dense Voxel	48.4
Huang et al. (2019)	TextureNet	(✓)[2]	Mesh	56.6
Jiang et al. (2019)	HPEIN	✗	Point graph	61.8
Hermosilla et al. (2018)	MCCNN	✗	Point cloud	63.3
Chiang et al. (2019)[1]	joint point-based	✓	Point cloud	63.4
Ours	MVPNet	✓	Point cloud	64.1
Wu et al. (2019b)[1]	PointConv	✗	Point cloud	66.6
Thomas et al. (2019)[1]	KP-FCNN	✗	Point cloud	68.4
Graham et al. (2018)	SparseConvNet	✗	Sparse Voxel	72.5
Choy et al. (2019)[1]	MinkowskiNet	✗	Sparse Voxel	73.6

[1] Published posterior to submission of our contributions.
[2] Samples 10x10 pixel surface texture patch from mesh for each point.

Table 4.3: Comparison of 2D-3D fusion and 3D only methods on ScanNet 3D semantic label benchmark (hidden test set). Note that the original PointNet++ results (Qi et al., 2017b) are very low. We have obtained much better results with our re-implementation which we report in Table 4.7.

mesh, while we take higher-quality original images as input. In summary, these results advocate for the use of continuous sparse point cloud as canonical 2D-3D fusion space, because it is precise and efficient.

Comparison to all Benchmark Entries. Table 4.3 lists more entries in the ScanNet benchmark, also showing methods that use point cloud only as input. Note that we only exclude some lower-ranked and unpublished methods. At the time of our submission, we outperformed all methods, except SparseConvNet (Graham et al., 2018) that uses sparse voxels with hash table indexing.

Since publication, several new works have been submitted. MinkowskiNet (Choy et al., 2019) has slightly improved (+1.1 mIoU) over SparseConvNet (Graham et al., 2018) by using a 3D variant of their 4D MinkowskiNet34 and can be seen as improved re-implementation of SparseConvNet. Additionally, two point-cloud only methods now outperform us. Wu et al. (2019b) use a continuous convolution definition and weight the kernel function with the inverse point density. Thomas et al. (2019) propose a point-based network with spatial kernels that are more powerful than the MLP-based point-wise convolutions of PointNet++ (Qi et al., 2017b). Additionally, they use an implementation that can collate point clouds with varying number of points in a single batch, similarly as Graham et al. (2018). In our implementation, we are constrained to sample a fixed number of points during training. Although it seems like an insignificant implementation detail, sampling point clouds with a fixed number of points while surface area varies can negatively impact training, as it leads to varying point cloud density to

Method	mIoU	batch size	train time	forward time/scene*	#parameters
SparseConvNet (light)	57.5	32	18h	0.194s	2.7M
SparseConvNet (heavy)	68.2	4	>12d	2.21s	30.1M
ResNet (2D)	-	32	8h	-	23M
MVPNet (3-view)	65.9	32	12h	2.22s	0.98M
MVPNet (5-view)	67.3	32	18h	3.35s	0.98M

* Preprocessing time not included.

Table 4.4: Runtime comparison with SparseConvNet on a GTX 1080Ti (11GB). The mIoU is reported on the validation set of ScanNet. Note that, in order to fit the heavy version of SparseConvNet into 11GB GPU memory, we needed to use a smaller batch size, leading to a lower score than in the benchmark in Table 4.3.

which networks are sensitive.

Another advantage of MVPNet is its robustness to low resolution point clouds as detailed in Section 4.4.4. This is relevant to robotic applications, where point clouds are always much sparser than images due to sensor limitations.

Runtime Analysis. The top scores of Choy et al. (2019) and Graham et al. (2018) are achieved with point cloud input only, very deep networks and GPUs with 24GB of RAM and 16GB, respectively. In order to fit ours into a single 11GB GPU, while jointly processing 2D and 3D data, we use rather light networks (ResNet34 based U-Net and PointNet++). We compare with SparseConvNet – the only published work that outperformed us at the time of our contribution – in terms of training/inference time and number of parameters in Table 4.4. Therefore, we used the publicly released code of SparseConvNet[2]. They provide two versions in their code, referred to as light and heavy in Table 4.4: a small U-Net with 5cm voxels that fits into 11GB GPU memory (light) and a large network with 2cm voxels that requires 16GB (heavy). The table shows that MVPNet is comparable to SparseConvNet regarding performance on the validation set. However, MVPNet is able to converge in 20 hours on a GTX 1080Ti (incl. pretrain of 2D encoder-decoder), while it takes 12 days for heavy SparseConvNet with GTX 1080Ti, or 4 days with a Tesla V100.

MVPNet also compares favorably in terms of runtime to other 2D-3D fusion methods. For example, 3DMV (Dai and Nießner, 2018) takes much longer to evaluate than ours, probably due to their column-wise evaluation (500 s/scene vs. 3.35 s/scene). This shows that it is challenging to make 2D-3D fusion efficient, but that our approach and implementation are rather competitive.

[2] https://github.com/facebookresearch/SparseConvNet

Qualitative Results. In Figure 4.10, we show qualitative results from our 3D baseline (PointNet++ taking only colored point cloud (XYZRGB) as input), our 2D baseline (predictions from our 2D network unprojected to 3D), and MVPNet (2D-3D fusion). MVPNet produces the best results and one can see that cues from 2D images especially improve classes with flat shapes, i.e. refrigerator, picture, curtain, which lack discriminative geometric cues. A failure case is depicted in Figure 4.11.

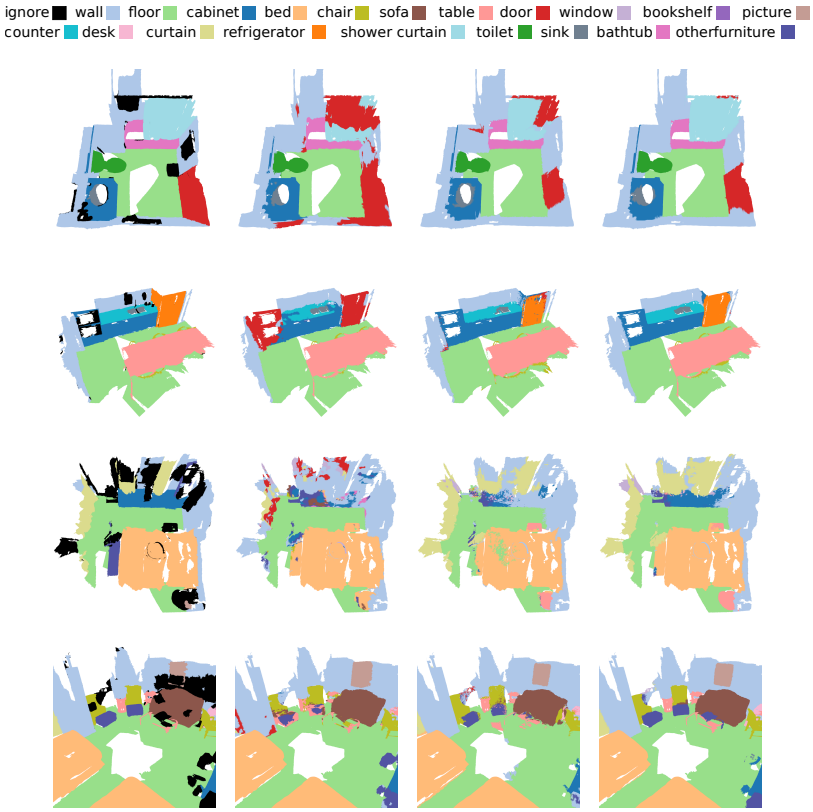
4.4.4 Robustness to Varying Point Cloud Density

Real-world 3D sensors such as LiDARs and depth cameras have much lower resolution than 2D RGB cameras and the point cloud density also varies with view point angle, lighting conditions, object-to-sensor distance and object surface reflectivity. Thus, it is important for algorithms to be robust to varying point cloud density at test time, because training data can hardly cover all cases. To examine robustness to sparsity, we uniformly subsample the whole scene point cloud and feed it to the networks that were trained on full resolution. We report the results in Figure 4.12. While our MVPNet is hardly affected at lower resolutions, the performance of the light version of SparseConvNet (Graham et al., 2018) suffers severely. Note that we only compare to the light version of SparseConvNet here due to our GPU limitations. Even though this leads to lower overall performance, we assume that the robustness to decreasing resolution behaves similarly as for the heavy version of SparseConvNet.

We attribute the performance difference between MVPNet and SparseConvNet mainly to two factors. First, the quality of our image features is not deteriorated when the point cloud is downsampled, because they are computed in the original dense 2D image. During 2D-3D lifting, even if few unprojected image pixels are finally used at coarse point cloud resolution, the image features can maintain their quality thanks to the receptive field of the 2D encoder-decoder. This is not the case for SparseConvNet where only the sparse RGB information at each 3D point is used. The second factor is related to the different neighborhood definition in voxel grids and point clouds. Voxel-based methods such as SparseConvNet have fixed neighbors defined by the discrete grid. Point-based methods on the other hand, use continuous locations and in each network layer neighbors are sampled, e.g. with ball query, that adapt naturally to the local point distribution. This enables our PointNet++ based approach to cope well with varying point cloud density.

4.5 Ablation Studies

To analyze our design choices and provide more insights, we conduct ablation studies on the validation set of ScanNet and report the mean IoU (mIoU)



Ground Truth PointNet++ XYZRGB Ours (2D) Ours (2D + 3D)

Figure 4.10: Qualitative results of 3D semantic segmentation for our 3D baseline (PointNet++), our 2D baseline (Multi-view 2D CNN) and MVP-Net which reach an mIoU of 57.9, 57.2 and 65.0, respectively. A common error mode of PointNet++ is to misclassify similarly shaped objects (shower curtain, refrigerator, etc.) as the most prevalent door category while MVP-Net succeeds.

Ground Truth PointNet++ XYZRGB Ours (2D) Ours (2D + 3D) color mesh (no input)

Figure 4.11: Failure case of our method for 3D semantic segmentation. On the right hand side in the image is an open door next to a bookshelf. Both have very similar "wooden" appearance and are spatially close which leads our method – which also relies on appearance information – to misclassify the bookshelf as door.

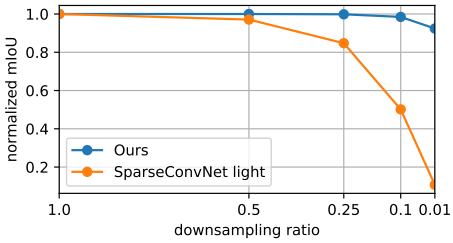


Figure 4.12: Robustness to input point cloud density of our method compared to the light version of SparseConvNet (Graham et al., 2018). Normalized mIOU equals 1.0 at full resolution (at downsampling ratio 1.0). The x-axis shows the ratio of points that are kept and the y-axis shows the mIoU on the validation set of ScanNet.

# views	1	3	5
Average coverage	68.1	92.9	**97.4**
mIoU	62.8	64.5	**64.8**

Table 4.5: Average coverage and mIoU as a function of the number of unprojected views. Results on validation set of ScanNet.

over 20 classes. The 2D encoder-decoder network is frozen in order to accelerate training, since we observe no significant improvement with end-to-end training. Unless stated otherwise, our 2D backbone is VGG16 (Simonyan and Zisserman, 2015). Our 3D backbone, PointNet++, contains 4 set abstraction (downsampling) and 4 feature propagation layers (upsampling). The numbers of centroids, points that are sampled at each layer, are 1024, 256, 64, 16 respectively.

4.5.1 Number of Views

We refer to *coverage* as the ratio of points in the input point cloud that have at least one unprojected neighbor point with image features at a distance less than 0.1m. Table 4.5 shows how the number of views affects coverage and mIoU. We removed the feature aggregation module for this experiment to get results that are independent from this design choice. With 1 view the coverage reaches 68.1%, and with 3 views already exceeds 90%. More views lead to higher mIoU, but introduce more computation. We choose 3 frames as default in this trade-off.

4.5.2 Feature Aggregation Module

In the following we study the parameters of our Feature Aggregation Module, defined in Equation 4.1, which distills features from the unprojected point

# views	k-nn	MLP	Aggregation	mIoU
1	1	w/o	none	61.7
1	3	w/o	sum	62.8
1	3	w/	sum	62.5
3	1	w/o	none	64.5
3	3	w/o	sum	64.5
3	3	w/	sum	**65.0**
3	3	w/	max	64.7

Table 4.6: Effect of feature aggregation. Results on validation set of ScanNet.

Method	mIoU
PointNet++ (XYZ) [baseline]	54.5
PointNet++ (XYZRGB) [baseline]	57.8
Multi-view 2D CNN	57.2
Ours (late fusion)	58.4
Ours (intermediate fusion)	64.8
Ours (early fusion)	**65.0**
Ours (w/o xyz)	62.8

Table 4.7: Effect of multiple modalities and different strategies of fusion. Results on validation set of ScanNet.

cloud $\mathcal{S}_{\text{dense}}$, obtained from multiple views, into the input point cloud $\mathcal{S}_{\text{sparse}}$. We report our results in Table 4.6 for 1 and 3 views, 1 or 3 nearest neighbors (k-nn) feature sampling, with or without MLP, and we also try maximum instead of sum as aggregation operation.

For the 1-view case, using 3 nearest neighbors instead of only 1 increases the performance by at least 0.8 mIoU. Due to the limited coverage of a single view, far-away image features are sampled for uncovered points and multiple neighbors might alleviate this problem by analyzing feature consistency between them.

For the 3-view case, we find that the number of nearest neighbors does not affect performance. This might be because coverage is already very high (92.9%) which means that, as opposed to the 1-view case, features can always be sampled from close-by. We also try summation instead of maximum to pool the features which does not significantly change the results. Using an MLP can slightly improve, maybe because it can transform 2D image features to an embedding space more consistent with the 3D representation. Our final choice for all other experiments is 3 nearest neighbors with MLP and sum aggregation.

4.5.3 Fusion

In this section we want to answer the question how to best fuse geometry and image features with point cloud based networks and give an insight about the strength of each modality. In Table 4.7 we report our quantitative results on the validation set. The point cloud only baseline, PointNet++, yields 54.5 mIoU with XYZ only and 57.8 mIoU with additional color information (XYZRGB).

In order to assess the strength of multi-view vs. geometry features, we conduct an experiment with 3 views where we unproject the output semantic labels of the pretrained 2D encoder-decoder to 3D and attribute the nearest neighbor 2D label to each 3D point in the point cloud. This multi-view 2D CNN approach can already achieve similar performance as PointNet++ on colored point clouds, which confirms the benefit of features computed on dense 2D images before 2D-3D lifting.

Moreover, we evaluate the three fusion strategies introduced in Section 4.3.5. We could yield slightly better performance with the late fusion approach (+1.2 mIoU) than with the 2D CNN baseline. Intermediate fusion leads to much better results (+7.6 mIoU) than late fusion. Early fusion can reach the best score (+7.8 mIoU), and uses fewer parameters and computation compared to intermediate fusion. The observation is different from the voxel-based method 3DMV (Dai and Nießner, 2018), where geometric features and image features are concatenated late, at roughly 2/3 in the network. We conclude, similar as in Chapter 3, that the optimal fusion strategy depends on the architecture.

Moreover, we investigate whether it is necessary to add geometric features (XYZ coordinates) in the early fusion approach or if the image features are sufficient. In fact, PointNet++ already induces a geometric hierarchy due to neighborhood sampling at each layer. Also, the unbalanced dimensions when concatenating the XYZ coordinates (dimension 3) with the image features (dimension 64) at the PointNet++ input could lead to a neglection of geometric features (XYZ coordinates). Nonetheless, the obtained result (-2.2 mIoU without XYZ) proves the contrary and indicates that MVPNet actually benefits from geometric features as complementary information to images.

4.5.4 Stronger Backbone

To investigate the effect of stronger 2D backbones, we replace VGG16 with VGG19 and ResNet34. Table 4.8 shows the results of 2D CNN baselines and our MVPNet with different backbones on the validation set. Intuitively, stronger backbones lead to higher mIoU, and ResNet34 performs best. Due to runtime performance we choose VGG16 as backbone for ablation experiments and ResNet34 for best performance.

Method	mIoU
2D CNN (VGG16)	57.2
2D CNN (VGG19)	58.3
2D CNN (ResNet34)	59.6
2D CNN (VGG16) + PointNet++(SSG)	65.0
2D CNN (VGG19) + PointNet++(SSG)	65.5
2D CNN (ResNet34) + PointNet++(SSG)	65.9
2D CNN (VGG16) + PointNet++(MSG)	65.0
2D CNN (VGG16) + PointNet++(SSG, more centroids)	**66.4**

Table 4.8: 2D CNN baselines and our MVPNet with different backbones. Results on validation set of ScanNet.

To test a stronger 3D backbone, we double the numbers of sampled centroids to (2048, 512, 128, 32), which increases the mIoU by 1.4. We also try to replace single-scale grouping (SSG) with multi-scale grouping (MSG) in PointNet++, but observe no improvement. As the image features already contain contextual information, it might not be necessary to process multiple scales explicitly with the MSG version.

4.6 Experiments on S3DIS

In order to validate our approach on a different dataset, we apply our pipeline to Stanford Indoor 3D (S3DIS) (Armeni et al., 2017).

4.6.1 S3DIS Dataset

Stanford Indoor 3D (S3DIS) is captured with the Matterport camera that is mounted on a tripod and collects RGB-D data at 360° by rotating its three structured light sensors that are mounted at three different angles (top, middle, bottom). The data was collected in three different buildings of educational and office use and the resulting scans are split floor-wise into six areas. Initially, the data only comprised 3D point clouds and classification ground truth (S3DIS), but was then amended with 2D images and xyz-maps in a second publication, named 2D-3D Semantics (2D-3D-S) (Armeni et al., 2017).

4.6.2 3D Semantic Segmentation

We evaluate MVPNet on S3DIS using the xyz maps from 2D-3D-S (Armeni et al., 2017). We use area 5 for testing and the rest for training. To simplify data processing and speed up training, we carry out a first pre-processing step, where we compute the 2D features and save them to disk. Therefore, we first compute the 2D features from 32 RGB-D views per room. The images are resized to 360x360. Then, the 2D feature maps are unprojected

Method	mIoU	mAcc	OA
Li et al. (2018)	57.26	63.86	85.91
Landrieu and Simonovsky (2018)	58.04	66.50	86.38
Wang et al. (2018)	58.27	67.01	-
Thomas et al. (2019)[1]	**67.1**	**72.8**	-
Qi et al. (2017b) [our implementation]	56.19	64.09	85.26
Ours	62.43	68.68	88.08

[1] Published posterior to submission of our contributions.

Table 4.9: Segmentation results on S3DIS Area 5. We report mIoU, mean over class accuracies (mAcc) and overall accuracy (OA).

to 3D and propagated to the nearest point in the point cloud. We take the mean of the 2D feature vectors at each 3D point before saving them to disk room-wise. The chunk-wise training procedure is the same as for ScanNet, described in Section 4.4.

The final score is obtained by ensembling an MVPNet and a PointNet++ (point cloud only) model. Results are shown in Table 4.9. At the time of contribution, we achieved above state-of-the-art performance, but since then, Thomas et al. (2019) have outperformed us. Unfortunately, the dataset is not as diverse as ScanNet. This leads to overfitting, especially when training the 2D network.

4.7 Conclusion

We have proposed a framework to fuse 2D multi-view images and 3D point clouds in an effective way by computing image features in 2D first, lifting them to 3D, and then fuse complementary geometry and image information in canonical 3D space. Extensive experiments are conducted on the ScanNet Semantic Segmentation benchmark which prove the advantage of fusing features from multiple local views into a global point cloud, because they help to obtain more informative point-wise features. Note that these MVPNet features could also benefit other tasks. One example is 3D instance semantic segmentation, a different task on the ScanNet benchmark, that is more precise than object detection: instead of regressing boxes, point masks which describe the exact shape of each object are to be predicted. Such an instance segmentation approach could operate on top of MVPNet features and preliminary experiments in our publication Jaritz et al. (2019) look promising.

While we outperform all other 2D-3D fusion methods, we do not reach the score of the best 3D point cloud only networks. We observe that very large models (Graham et al., 2018; Choy et al., 2019) and implementations that collate whole scenes with a varying number points into one batch (Graham et al., 2018; Choy et al., 2019; Thomas et al., 2019) perform par-

ticularly well. In our current implementation the number of input points must be fixed to collate multiple examples into a batch. To test whole scene processing, we sample a fixed number of 32k points in the input point cloud. This leads to low point density in large scenes and high point density in small scenes which can negatively influence performance as neural networks are sensitive to varying sparsity. Although extending to a varying number of points is more complicated to implement in the 2D-3D multi-view setting than when using 3D only, it could be integrated in the future to improve results.

Like in Chapter 3, we found that sparsity plays a crucial role. We show in our robustness analysis in Section 4.4.4 that the lower the point cloud resolution, the more important multi-view fusion becomes, because dense 2D images can compensate for point cloud sparsity. In ScanNet, the point clouds are rather dense. This is natural, because the whole scene point cloud is obtained by reconstruction from a highly redundant RGB-D sequence. In outdoor settings on the other hand, point clouds are typically much sparser compared to image resolution. In the next chapter, we carry out 2D-3D fusion in the outdoor scenario and draw from our experience gained on 3D semantic segmentation in this chapter.

Adaptation de domaine non supervisée inter-modalités 2D-3D

French Summary of the Chapter "2D-3D Cross-modal Unsupervised Domain Adaptation"

Dans ce chapitre, nous introduisons la nouvelle tâche *d'adaptation de domaine non supervisée inter-modalités* où on a accès à des données multi-capteurs dans une base de données source annotée et une base cible non annotée. Nous proposons une méthode d'apprentissage inter-modalités 2D-3D via une imitation mutuelle entre les réseaux d'images et de nuages de points pour résoudre l'écart de domaine source-cible. Nous montrons en outre que notre méthode est complémentaire à la technique unimodale existante dite de pseudo-labeling.

Chapter 5

2D-3D Cross-modal Unsupervised Domain Adaptation

The contributions of this chapter were published in (Jaritz et al., 2020):

Jaritz, M., Vu, T.-H., de Charette, R., Wirbel, E., and Pérez, P. (2020). xMUDA: Cross-Modal Unsupervised Domain Adaptation for 3D semantic segmentation. *CVPR 2020*.

We plan to publish the code and dataset splits.

Contents

5.1 Introduction **104**
5.2 Related Work **106**
5.3 xMUDA **110**
5.3.1 Architecture 111
5.3.2 Learning Scheme 112
5.4 Experiments **114**
5.4.1 Datasets 114
5.4.2 Implementation Details 115
5.4.3 Main Experiments 116
5.4.4 Extension to Fusion 118
5.5 Ablation Studies **121**
5.5.1 Segmentation Heads 121
5.5.2 Cross-modal Learning on Source 122
5.5.3 Cross-modal Learning for Oracle Training 122
5.6 Conclusion **123**

5.1 Introduction

In Chapter 4, we investigated how to improve 3D semantic segmentation through 2D-3D fusion. In this chapter, we want to build on this, but now focus on exploiting 2D-3D modalities to address the problem of Unsupervised Domain Adaptation (UDA).

Annotating a point cloud for 3D semantic segmentation is particularly tedious and took up to 4.5 hours for a 100m by 100m birdview tile for the SemanticKITTI dataset (Behley et al., 2019), because the annotator needed to inspect the scene from multiple points of view. As this labeling process is so costly, we can oftentimes not afford the annotation of a new dataset. Alternatively, we can train a model on a similar, labeled *source* dataset and then deploy this model on our unlabeled *target* dataset. Let us consider that the source dataset was collected during the day and the target dataset during the night. In this case, we will encounter a significant performance drop of the model's test performance when switching from day to night, because the daylight images seen during training are visually very different from the dark night images. In other words, we encounter a *domain shift* between the distributions of the day *source domain* and night *target domain* from which the two datasets were sampled respectively. The goal of *unsupervised* domain adaptation is to improve our model's performance on the *unlabeled* target set from which unannotated samples are available at train time.

Existing work in domain adaptation concerns mostly 2D semantic segmentation (Hoffman et al., 2018; Zou et al., 2019; Vu et al., 2019a; Li et al., 2019) and rarely 3D (Wu et al., 2019a). We also observe that previous domain adaptation work focuses on single modality, whereas 3D datasets are often multi-modal, consisting of 3D point clouds *and* 2D images. While the complementarity between these two modalities is already exploited by both human annotators and learned models to localize objects in 3D scenes (Liang et al., 2018; Qi et al., 2018), we consider it through a new angle, asking the question: If 3D and 2D data are available in the source *and* target domain, can we capitalize on multi-modality to address Unsupervised Domain Adaptation (UDA)?

We coin our method *cross-modal UDA*, 'xMUDA' in short, and consider three real-to-real adaptation scenarios with different lighting conditions (day-to-night), environments (country-to-country) and sensor setups (dataset-to-dataset). It is a challenging task for various reasons. The heterogeneous input spaces (2D and 3D) make the pipeline complex as it implies to work with heterogeneous network architectures and 2D-3D projections. In fusion, if two sensors register the same scene, there is shared information between both, but each sensor also has private (or exclusive) information. One modality can be stronger than the other in a certain case, but it can be the other way around in another, depending on class, context, resolution, etc. This makes selecting the "best" sensor based on prior knowledge

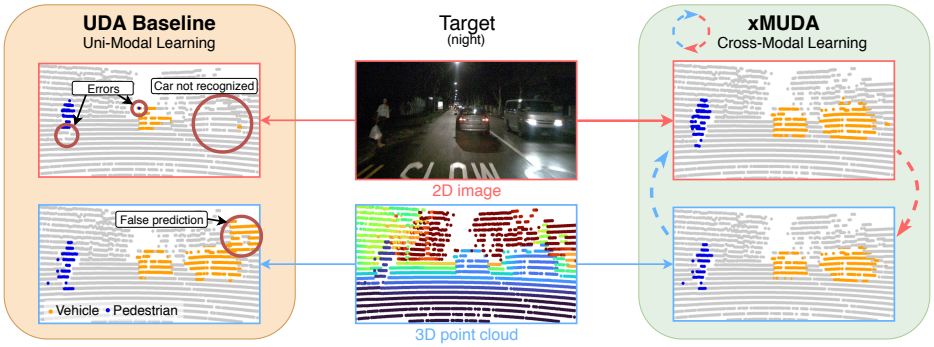


Figure 5.1: Advantages of cross-modal UDA (xMUDA) in presence of domain gap (day-to-night). On this 3D semantic segmentation example, the UDA baseline on the left side of the figure (Li et al., 2019) does not detect the car on the right in the 2D camera image due to the day/night domain shift. With xMUDA on the right side of the figure, 2D learns the appearance of cars in the dark from information exchange with the 3D LiDAR point cloud, and 3D learns to reduce false predictions.

unfeasible. Additionally, each modality can be affected differently by the domain shift. For example, camera is deeply impacted by the day-to-night domain change, while LiDAR is relatively robust to it, as shown on the left in Figure 5.1.

In order to address these challenges, we propose the xMUDA framework where information can be exchanged between 2D and 3D in order to learn from each other for UDA (see right side of Figure 5.1). We use a disentangled 2-stream architecture to address the domain gap individually in each modality. Our learning scheme allows robust balancing of the segmentation and cross-modal objectives. In addition, xMUDA is complementary to self-training with pseudo-labels (Li et al., 2019), a popular UDA technique, as it exploits a different source of knowledge. Finally, it is common practice to use feature fusion (e.g., early or late fusion) when multiple modalities are available (Hazirbas et al., 2016; Valada et al., 2019; Liang et al., 2018): our framework can be extended to fusion while maintaining a disentangled cross-modal objective.

Our contributions can be summarized as follows:

- We define new UDA scenarios and propose corresponding splits on recently published 2D-3D datasets.
- We design an architecture that enables cross-modal learning by disentangling private and shared information in 2D and 3D.
- We propose a novel UDA learning scheme where modalities can learn from each other in balance with the main objective. It can be ap-

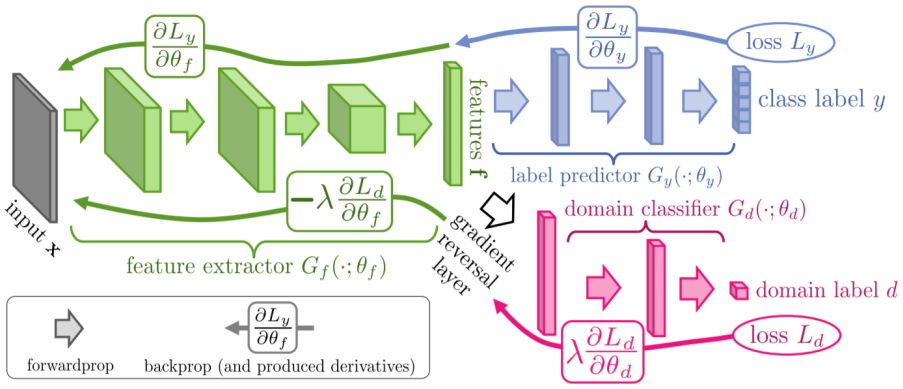


Figure 5.2: Adversarial training is achieved with the gradient reversal layer, where the gradient to train the domain classifier, which discriminates between source and target, is multiplied by -1, so that the gradient has the opposite effect on the feature extractor: to produce features that are domain-invariant. Source: Ganin et al. (2016).

plied on top of state-of-the-art self-training techniques to boost performance.

- We showcase how our framework can be extended to late fusion and produce superior results.

On the different proposed benchmarks we outperform the single-modality state-of-the-art UDA techniques by a significant margin. Thereby, we show that the exploitation of multi-modality for UDA is a powerful tool that can benefit a wide range of multi-sensor applications.

5.2 Related Work

Unsupervised Domain Adaptation (UDA). The past few years have seen an increasing interest in unsupervised domain adaptation techniques applied to image-based tasks such as classification and semantic segmentation. The initial intuition behind those methods is the following. As the classifier (or segmentor) can only be trained on the source dataset where labels are available, we could enforce the features that are fed to this classifier to be domain-invariant, such that the classifier can also perform well on the target dataset, i.e. the feature extractor should be trained such that source and target features are aligned. Examples to measure the distance between source and target feature distributions that have been used in the context of UDA are maximum mean discrepancy (MMD) (Tzeng et al., 2014; Long et al., 2015b) and covariance (Sun and Saenko, 2016). In the same spirit, Ganin et al. (2016) propose adversarial training, depicted in Figure 5.2,

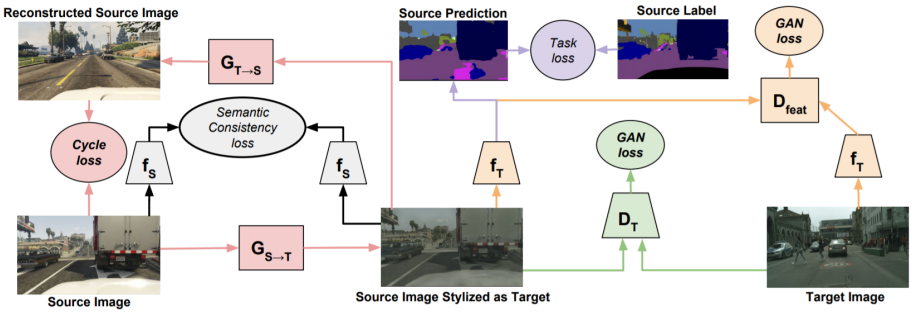


Figure 5.3: CyCADA (Hoffman et al., 2018) translates images from source to target to be able to use the labels in the target domain. In the following we summarize the losses. Cycle consistency (red) is enforced on the mapping from source to target and back to source to train the generators $G_{S\to T}$ and $G_{T\to S}$. Adversarial loss is applied on image (green) and feature-level (orange). Semantic consistency is imposed between the features of the source image and its mapped target version (black). Finally, the task loss is used to train for semantic segmentation (purple). Source: Hoffman et al. (2018).

where a 'domain classifier' predicts if the features originate from a source or target example. The gradient to train the domain classifier to discriminate correctly between source and target is multiplied by -1 in the 'gradient reversal layer' to train the feature extractor with the opposite objective, i.e. to produce domain-invariant features that can not be distinguished by the domain classifier.

The methods introduced above exclusively focus on the classification task. In order to extend UDA to semantic segmentation, Hoffman et al. (2016) employ a fully-convolutional network (FCN) architecture and carry out region-wise domain-adversarial adaptation. Instead of using gradient reversal from Ganin et al. (2016), Hoffman et al. (2016) adopt the Generative Adversarial Net (GAN) approach (Goodfellow et al., 2014), where the feature extractor and domain discriminator are trained in an alternating procedure with two different losses. While the first loss optimizes the discriminator to correctly classify domains, the second loss trains the feature extractor with the objective of confusing the discriminator.

The work described so far aligns source-target distributions in *feature-space*. A different line of work explores alignment in *input-space*, where source images are translated to the target domain (Hoffman et al., 2018; Liu et al., 2017; Murez et al., 2018; Wu et al., 2018b).

Feature-space alignment is limited, because it does not enforce semantic consistency. For example, the target features of a cat can be mapped to source features of a dog. CyCADA (Hoffman et al., 2018), a representative example of input-space alignment methods, shown in Figure 5.3, addresses this limitation by promoting cycle-consistency: the generator $G_{S\to T}$ trans-

lates a source image to target, and a second generator $G_{T\to S}$ translates an image from target back to source. The cycle-consistency loss is then applied between the original source image I_S and the back and forth mapped source image $G_{S\to T}(G_{T\to S}(I_S))$. Additionally, the adversarial (GAN) losses enforce alignment on image-level with discriminator D_T (green) and feature-level with discriminator D_{feat} (orange). The feature extractors f_S and f_T operate on source and target, respectively, and f_S is pretrained and fixed, used to impose semantic consistency (black). Finally, the task loss optimizes for the prediction of the final semantic segmentation.

While image-translation methods allow for visual inspection of the adaptation quality, they have since been outperformed by adversarial approaches that align source-target distributions in *output-space* (Tsai et al., 2018; Vu et al., 2019a). These techniques take scene structure into account while carrying out alignment which benefits the highly structured segmentation task.

Curriculum learning (Zhang et al., 2017) is a different approach where simple things are learned first, such as global label distributions over images.

Revisited from semi-supervised learning (Lee, 2013), self-training with pseudo-labels has recently been proven effective for UDA (Li et al., 2019; Zou et al., 2019). In a first step, a model is trained only on the source dataset and then used without adaptation to generate predictions on the target dataset. These target predictions are called pseudo-labels. Li et al. (2019) refine the pseudo-labels by discarding less confident ones (below the class-wise median threshold), where the confidence is measured as probability of the most probable class. Zou et al. (2019) investigate different confidence regularizers to improve pseudo-label quality. In a second step, the pseudo-labels are used to 'self-train' the model on the target dataset. In this work, we employ self-training with pseudo-labels and label refinement as proposed by Li et al. (2019). Note that it is also possible to achieve pseudo-labeling in one-stage training as recently done by Pizzati et al. (2020).

While most existing works consider UDA in the 2D world, very few tackle the 3D counterpart. Wu et al. (2019a) adopted activation correlation alignment (Morerio et al., 2018) for UDA in 3D segmentation from LiDAR point clouds. In this work, we investigate the same task, but differently: our system operates on multi-modal input data, i.e., RGB + LiDAR.

To the best of our knowledge, there are no previous UDA works in 2D/3D semantic segmentation for multi-modal scenarios. Only some consider the extra modality, i.e. depth, solely available at training time on source domain and leverage such *privileged information* to boost adaptation performance (Lee et al., 2019; Vu et al., 2019b). As depicted in Figure 5.4, in (Vu et al., 2019b) the depth serves as ground truth to train a subnetwork to predict the depth from RGB, trained supervisedly on the source dataset. The depth aware features from the subnetwork are then fused with the main features, after which the final layer to predict the semantic segmentation fol-

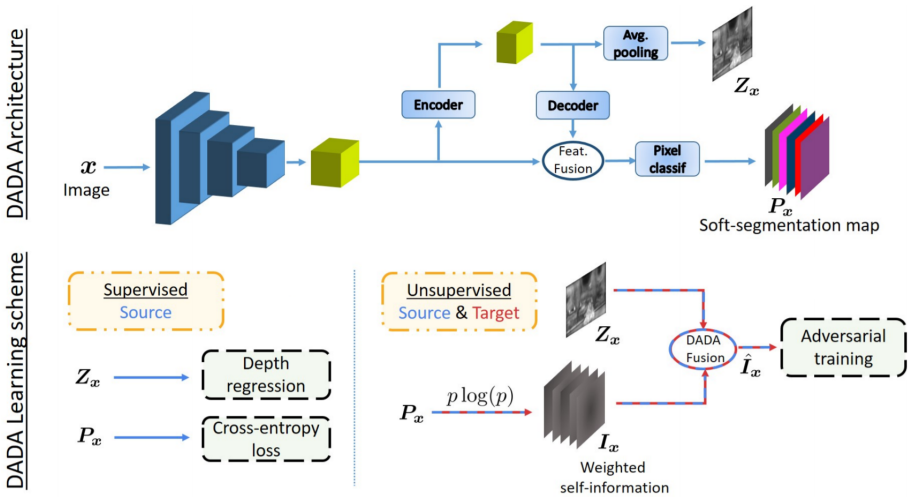


Figure 5.4: Vu et al. (2019b) use depth as *privileged information*. Specifically, as shown in the architecture on top, the depth is predicted by a small subnetwork and trained in a supervised fashion on the source dataset. The subnetwork produces depth-aware features that are then fused with the main features, followed by the segmentation layer. The depth ground truth is supposed to be absent in the target dataset and can thus not be used for unsupervised learning. Source: Vu et al. (2019b) (modified).

lows. This approach seems to regularize the output and benefit performance on the target domain. However, in many datasets the extra modality (e.g. depth) is also available in the target domain and Vu et al. (2019b) do not exploit this fact. In this chapter, we assume all modalities to be available at train and test time on both source and target domains. This allows the usage of the additional modality as independent input, as opposed to using it only as privileged information, and thereby exploits multi-modality as unsupervised learning signal for domain adaptation.

Multi-Modality Learning. In a supervised setting, performance can naturally be improved by fusing features from multiple sources. RGB-Depth fusion is a geometrically simple case as there is dense pixel-to-pixel correspondence (Hazirbas et al., 2016; Valada et al., 2019). It is harder to fuse a 3D point cloud with a 2D image, because they live in different metric spaces. In Chapter 3, we projected the point cloud into 2D camera space for the task of RGB-guided depth completion. Another solution is to project 2D and 3D features into a 2D 'bird eye view' for object detection (Liang et al., 2018). In Chapter 4, we lift 2D features from multi-view images to the 3D point cloud to enable joint 2D-3D processing for 3D semantic segmentation. Here, we share the same goal of 3D semantic segmentation as in Chapter 4. However, we focus on how to exploit multi-modality for UDA instead of

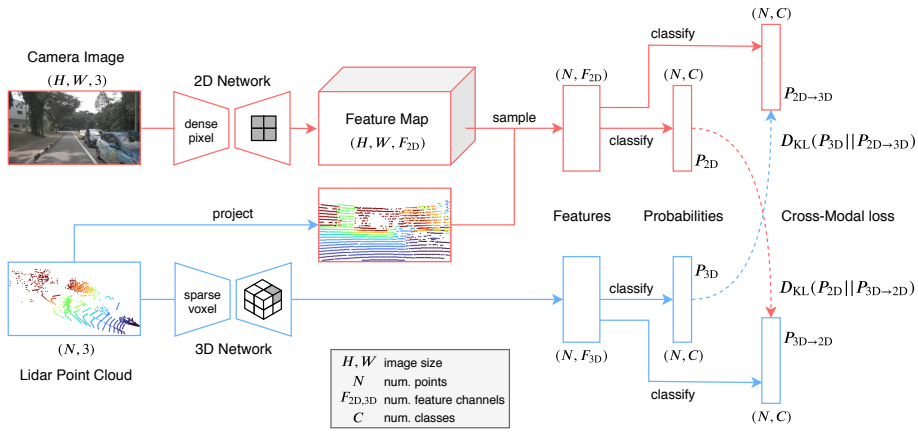


Figure 5.5: Overview of our xMUDA framework for 3D semantic segmentation. The architecture comprises a 2D stream which takes an image as input and uses a U-Net-style 2D ConvNet (Ronneberger et al., 2015), and a 3D stream which takes the point cloud as input and uses a U-Net-Style 3D SparseConvNet (Graham et al., 2018). Feature outputs of both streams have same length N, equal to the number of 3D points. To achieve that, we project the 3D points into the image and sample the 2D features at the corresponding pixel locations. The 4 segmentation outputs consist of the main predictions P_{2D}, P_{3D} and the mimicry predictions $P_{2D\to3D}, P_{3D\to2D}$. We transfer knowledge across modalities using KL divergence, $D_{KL}(P_{3D}||P_{2D\to3D})$, where the objective of the 2D mimicry head is to estimate the main 3D output and vice versa, $D_{KL}(P_{2D}||P_{3D\to2D})$.

supervised learning and only use single view images from the front camera and the corresponding point clouds.

5.3 xMUDA

The aim of cross-modal UDA (xMUDA) is to enable information exchange between modalities so that they can learn from each other in a way that benefits domain adaptation. This is difficult to investigate with the fusion architectures presented in Chapters 3 and 4, because they pick the most informative features of each modality with regard to optimizing an overall objective (e.g. the segmentation loss) rather than explicitly modeling the interaction between the modalities. In this chapter, we therefore focus on a co-learning approach where two independent networks are trained jointly; and we define a loss between those networks to explicitly model cross-modal learning. Specifically, we let the two networks mutually mimic each other's outputs, so that they can both benefit from their counterpart's strengths.

We investigate xMUDA using point cloud (3D modality) and image (2D

modality) on the task of 3D semantic segmentation. An overview is depicted in Figure 5.5. We first describe the architecture in Section 5.3.1, our learning scheme in Section 5.3.2. After having validated our xMUDA framework in the co-learning setup with two independent 2D/3D predictions, we also showcase how xMUDA can be extended to fusion networks with a single fused prediction.

In the following, we consider a *source* dataset $\mathcal{S}$, where each sample consists of 2D image $\boldsymbol{x}_s^{\text{2D}}$, 3D point cloud $\boldsymbol{x}_s^{\text{3D}}$ and 3D segmentation labels $\boldsymbol{y}_s^{\text{3D}}$ as well as a *target* dataset $\mathcal{T}$, lacking annotations, where each sample only consists of image $\boldsymbol{x}_t^{\text{2D}}$ and point cloud $\boldsymbol{x}_t^{\text{3D}}$. Images $\boldsymbol{x}^{\text{2D}}$ are of spatial size $(H, W, 3)$ and point clouds $\boldsymbol{x}^{\text{3D}}$ of spatial size $(N, 3)$, where N is the number of 3D points within the camera field of view.

5.3.1 Architecture

To allow cross-modal learning, it is crucial to extract features specific to each modality. Opposed to 2D-3D architectures where 2D features are lifted to 3D (Liang et al., 2018), we use a 2-stream architecture with independent 2D and 3D branches that do *not* share features (see Figure 5.5).

In this work, we select SparseConvNet (Graham et al., 2018) as 3D network, because, as we have seen in Chapter 4, SparseConvNet is the state-of-the-art on 3D semantic segmentation on the ScanNet benchmark (Dai et al., 2017). As expected, we could confirm the superiority of SparseConvNet over PointNet++ (Qi et al., 2017b) on the driving dataset NuScenes (Graham et al., 2018). For more details, please refer to the introduction of related work in the preceding chapter (cf. Section 4.2).

As 2D network, we employ a modified version of U-Net (Ronneberger et al., 2015) with ResNet34 (He et al., 2016) as already proposed in Chapter 4.

Even though each stream has a specific network architecture, it is important that the outputs are of same size to allow cross-modal learning. We provide implementation details in Section 5.4.2.

Dual Segmentation Head. We call segmentation head the last linear layer in the network that transforms the output features into logits followed by a softmax function to produce the class probabilities. For xMUDA, we establish a link between 2D and 3D with a 'mimicry' loss between the output probabilities, i.e., each modality should predict the other modality's output. This allows us to explicitly control the cross-modal learning.

In a naive approach, each modality has a single segmentation head and a cross-modal optimization objective aligns the outputs of both modalities. Unfortunately, this leads to only using information that is shared between the two modalities, while discarding private information that is exclusive to each sensor (more details in the ablation study in Section 5.5.1). This

is an important limitation, as we want to leverage both private and shared information, in order to obtain the best possible performance.

To preserve private information while benefiting from shared knowledge, we introduce an additional segmentation head to uncouple the mimicry objective from the main segmentation objective. This means that the 2D and 3D streams both have two segmentation heads: one main head for the best possible prediction, and one mimicry head to estimate the other modality's output.

The outputs of the 4 segmentation heads (see Figure 5.5) are of size (N, C), where C is equal to the number of classes such that we obtain a vector of class probabilities for each 3D point. The two main heads produce the best possible predictions, P_{2D} and P_{3D} respectively for each branch. The two mimicry heads estimate the other modality's output: 2D estimates 3D ($P_{\text{2D}\to\text{3D}}$) and 3D estimates 2D ($P_{\text{3D}\to\text{2D}}$).

5.3.2 Learning Scheme

The goal of our cross-modal learning scheme is to exchange information between the modalities in a controlled manner to teach them to be aware of each other. This auxiliary objective can effectively improve the performance of each modality and does not require any annotations which enables its use for UDA on target dataset $\mathcal{T}$. In the following we define the basic supervised learning setup, our cross-modal loss $\mathcal{L}_{\text{xM}}$, and the additional pseudo-label learning method. The loss flows are depicted in Figure 5.6.

Supervised Learning. The main goal of 3D segmentation is learned through cross-entropy in a classical supervised fashion on the source data. We can write the segmentation loss $\mathcal{L}_{\text{seg}}$ for each network stream (2D and 3D) as:

$$\mathcal{L}_{\text{seg}}(\boldsymbol{x}_s, \boldsymbol{y}_s^{\text{3D}}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} \boldsymbol{y}_s^{(n,c)} \log \boldsymbol{P}_{\boldsymbol{x}_s}^{(n,c)}, \tag{5.1}$$

where $\boldsymbol{x}_s$ is either $\boldsymbol{x}_s^{\text{2D}}$ or $\boldsymbol{x}_s^{\text{3D}}$.

Cross-modal Learning. The objective of unsupervised learning across modalities is twofold. Firstly, we want to transfer knowledge from one modality to the other on the target dataset. For example, let one modality be sensitive and the other more robust to the domain shift, then the robust modality should teach the sensitive modality the correct class in the target domain where no labels are available. Secondly, we want to design an auxiliary objective on source and target, where the task is to estimate the other modality's prediction. By mimicking not only the class with maximum probability, but the whole distribution, more information is exchanged, leading

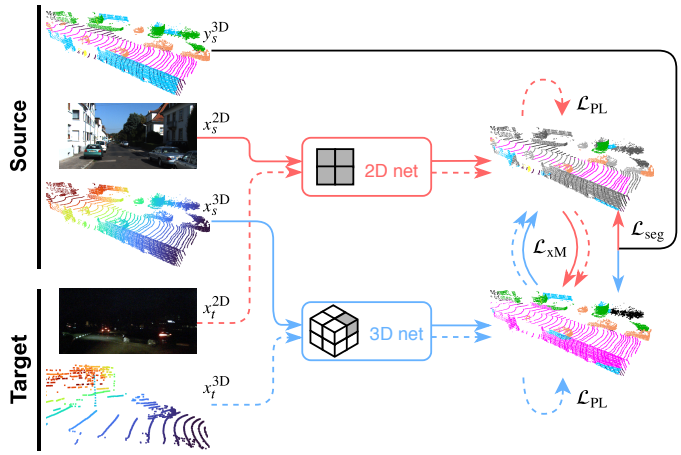


Figure 5.6: Proposed UDA setup. xMUDA learns from supervision on the source domain (plain lines) and self-supervision on the target domain (dashed lines), while benefiting from the cross-modal predictions of 2D/3D modalities.

to softer labels. As shown by Zou et al. (2019) soft labels perform better than their hard-label counterpart.

In order to achieve both objectives, we choose KL divergence to align the mimicking probability distribution of one modality with the truly predicted probability distribution of the other modality. We define the cross-modal loss $\mathcal{L}_{\text{xM}}$ as follows:

$$\mathcal{L}_{\text{xM}}(\boldsymbol{x}) = \boldsymbol{D}_{\text{KL}}(\boldsymbol{P}_{\boldsymbol{x}}^{(n,c)} || \boldsymbol{Q}_{\boldsymbol{x}}^{(n,c)}) \tag{5.2}$$

$$= -\frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} \boldsymbol{P}_{\boldsymbol{x}}^{(n,c)} \log \frac{\boldsymbol{P}_{\boldsymbol{x}}^{(n,c)}}{\boldsymbol{Q}_{\boldsymbol{x}}^{(n,c)}}, \tag{5.3}$$

with $(\boldsymbol{P}, \boldsymbol{Q}) \in \{(\boldsymbol{P}_{\text{2D}}, P_{\text{3D}\to\text{2D}}), (\boldsymbol{P}_{\text{3D}}, P_{\text{2D}\to\text{3D}})\}$ where $\boldsymbol{P}$ is the target distribution from the main prediction which is to be estimated by the mimicking prediction $\boldsymbol{Q}$. This loss is applied on the source and the target domain as it does not require ground truth labels and is the key to our proposed domain adaptation framework. For source, $\mathcal{L}_{\text{xM}}$ can be seen as an auxiliary mimicry loss in addition to the main segmentation loss $\mathcal{L}_{\text{seg}}$.

The complete optimization objective for each network stream (2D and 3D) is the combination of the segmentation loss $\mathcal{L}_{\text{seg}}$ on source and the cross-modal loss $\mathcal{L}_{\text{xM}}$ on source and target:

$$\min_{\theta} \left[\frac{1}{|\mathcal{S}|} \sum_{\boldsymbol{x}_s \in \mathcal{S}} \left(\mathcal{L}_{\text{seg}}(\boldsymbol{x}_s, \boldsymbol{y}_s^{\text{3D}}) + \lambda_s \mathcal{L}_{\text{xM}}(\boldsymbol{x}_s) \right) + \frac{1}{|\mathcal{T}|} \sum_{\boldsymbol{x}_t \in \mathcal{T}} \lambda_t \mathcal{L}_{\text{xM}}(\boldsymbol{x}_t) \right], \tag{5.4}$$

where λ_s, λ_t are hyperparameters to weight $\mathcal{L}_{\text{xM}}$ on source and target respectively and θ are the network weights of either the 2D or the 3D stream.

There are parallels between the cross-modal learning and model distillation which also adopts KL divergence as mimicry loss, but with the goal to transfer knowledge from a large network to a smaller one in a supervised setting (Hinton et al., 2014). Recently Zhang et al. (2018) introduced Deep Mutual Learning where an ensemble of uni-modal networks are jointly trained to learn from each other in collaboration. Though to some extent, our cross-modal learning is of similar nature as those strategies, we tackle a different distillation angle, i.e. across modalities (2D/3D) and not in the supervised, but in the UDA setting.

Self-training with Pseudo-Labels. Cross-modal learning is complementary to the pseudo-labeling strategy (Lee, 2013) used originally in semi-supervised learning and recently in UDA (Li et al., 2019; Zou et al., 2019).

Once we have optimized a model with Equation 5.4 on the source dataset, we generate pseudo-labels offline for the whole target dataset. We then select highly confident labels with the thresholding method proposed by Li et al. (2019). Precisely, in a loop over all classes, we gather all the predictions where the current class was the most probable one and extract the probabilities of that class. Then, we compute the median value over all gathered probabilities. The median value is used as threshold below which the labels of that class are discarded, assumed not to be confident enough. This process is repeated for each class.

After that, a second training is carried out, initializing the network weights from scratch, using the produced pseudo-labels for an additional segmentation loss on the target training set. Consequently, the full optimization problem writes:

$$\min_{\theta} \Bigg[\frac{1}{|\mathcal{S}|} \sum_{\boldsymbol{x}_s} \Big(\mathcal{L}_{\text{seg}}(\boldsymbol{x}_s, \boldsymbol{y}_s^{\text{3D}}) + \lambda_s \mathcal{L}_{\text{xM}}(\boldsymbol{x}_s) \Big) + \frac{1}{|\mathcal{T}|} \sum_{\boldsymbol{x}_t} \Big(\lambda_{\text{PL}} \mathcal{L}_{\text{seg}}(\boldsymbol{x}_t, \hat{\boldsymbol{y}}^{\text{3D}}) + \lambda_t \mathcal{L}_{\text{xM}}(\boldsymbol{x}_t) \Big) \Bigg], \quad (5.5)$$

where λ_{PL} is weighting the pseudo-label segmentation loss and $\hat{\boldsymbol{y}}^{\text{3D}}$ are the pseudo-labels.

5.4 Experiments

5.4.1 Datasets

To evaluate xMUDA, we identified three real-to-real adaptation scenarios. In the *day-to-night* case, LiDAR has a small domain gap, because it is an active sensing technology, sending out laser beams which are mostly invariant to lighting conditions. In contrast, camera has a large domain gap as its passive

sensing suffers from lack of light sources, leading to drastic changes in object appearance. The second scenario is *country-to-country* adaptation, where the domain gap can be larger for LiDAR or camera: for some classes the 3D shape might change more than the visual appearance or vice versa. The third scenario, *dataset-to-dataset*, comprises changes in the sensor setup, such as camera optics, but most importantly a higher LiDAR resolution on target. 3D networks are sensitive to varying point cloud density, as shown in the robustness study in Section 4.4.4, and the image could help to guide and stabilize adaptation.

We leverage recently published autonomous driving datasets nuScenes (Caesar et al., 2019), A2D2 (Geyer et al., 2019) and SemanticKITTI (Behley et al., 2019) in which LiDAR and camera are synchronized and calibrated allowing to compute the projection between a 3D point and its corresponding 2D image pixel. The chosen datasets contain 3D annotations. For simplicity and consistency across datasets, we only use the front camera image and the LiDAR points that project into it.

For nuScenes, the annotations are 3D bounding boxes and we obtain the point-wise labels for 3D semantic segmentation by assigning the corresponding object label if a point lies inside a 3D box; otherwise the point is labeled as background. We use the meta data to generate the splits for two UDA scenarios: Day/Night and USA/Singapore.

A2D2 and SemanticKITTI provide segmentation labels. For UDA, we define 10 shared classes between the two datasets. The LiDAR setup is the main difference: in A2D2, there are three LiDARs with 16 layers which generate a rather sparse point cloud and in SemanticKITTI, there is one high-resolution LiDAR with 64 layers.

We provide more details about the data splits in Appendix C.1.

5.4.2 Implementation Details

2D Network. We use a modified version of U-Net (Ronneberger et al., 2015) with a ResNet34 (He et al., 2016) encoder where we add dropout after the 3rd and 4th layer and initialize with ImageNet pretrained weights provided by PyTorch. In the decoder, each layer consists of a transposed convolution, concatenation with encoder features of same resolution (skip connection) and another convolution to mix the features. The network takes an image $\boldsymbol{x}^{\text{2D}}$ as input and produces an output feature map with equal spatial dimensions (H, W, F_{2D}), where F_{2D} is the number of feature channels. In order to lift the 2D features to 3D, we sample them at sparse pixel locations where the 3D points project into the feature map, in the same way as in MVPNet in Chapter 4, and obtain the final two-dimensional feature matrix (N, F_{2D}).

3D Network. For SparseConvNet (Graham et al., 2018) we leverage the official PyTorch implementation[1]. We use a U-Net architecture with 6 times downsampling and a voxel size of 5cm which is small enough to have only one 3D point per voxel.

Training. For data augmentation we employ horizontal flipping and color jitter in 2D, and x-axis flipping, scaling and rotation in 3D. Due to the wide angle image in SemanticKITTI, we crop a fixed size rectangle randomly on the horizontal image axis to reduce memory needs during training. There is severe class imbalance in nuScenes as most of the points do not belong to any object and are in consequence labeled with the background class. We address this problem using log-smoothed class frequencies as weights in the cross-entropy loss in all experiments. For the KL divergence for the cross-modal loss in PyTorch, we *detach* the target variable to only backpropagate in either the 2D or the 3D network. We use a batch size of 8, the Adam optimizer with $\beta_1 = 0.9, \beta_2 = 0.999$, and an iteration based learning schedule where the learning rate of 0.001 is divided by 10 at 80k and 90k iterations; the training finishes at 100k. We jointly train the 2D and 3D stream and at each iteration, accumulate gradients computed on source and target batch. All trainings fit into a single GPU with 11GB RAM.

There are two training stages. First, we train with Equation 5.4, where we apply the segmentation loss using ground truth labels on source and cross-modal loss on source *and* target. Once trained, we generate pseudo-labels as in (Li et al., 2019) from the last model. Note, that we do not select the best weights on the validation set, but rather use the last checkpoint to generate the pseudo-labels in order to prevent any supervised learning signal. In the second training with objective of Equation 5.5, an additional segmentation loss on target using the pseudo-labels is added; the networks weights are reinitialized from scratch. The 2D and 3D network are trained jointly and optimized on source and target at each iteration.

5.4.3 Main Experiments

We evaluate xMUDA on the three proposed cross-modal UDA scenarios and compare against a state-of-the-art uni-modal UDA method (Li et al., 2019).

We report mean Intersection over Union (mIoU) results for 3D segmentation in Table 5.1 on the target test set for the three UDA scenarios. We evaluate on the test set using the checkpoint that achieved the best score on the validation set. In addition to the scores of the 2D and 3D model, we show the ensembling result ('softmax avg') which is obtained by taking the mean of the predicted 2D and 3D probabilities after softmax. While the baseline is trained on source only without UDA and represents the lower bound in

[1] https://github.com/facebookresearch/SparseConvNet

Method	Day/Night			USA/Singapore			A2D2/SemanticKITTI		
	2D	3D	softmax avg	2D	3D	softmax avg	2D	3D	softmax avg
Baseline (source only)	42.2	41.2	47.8	53.4	46.5	61.3	36.0	36.6	41.8
UDA Baseline (PL) (Li et al., 2019)	43.7	45.1	48.6	55.5	51.8	61.5	37.4	44.8	47.7
xMUDA w/o PL	46.2	44.2	50.0	59.3	52.0	62.7	36.8	43.3	42.9
xMUDA	**47.1**	**46.7**	**50.8**	**61.1**	**54.1**	**63.2**	**43.7**	**48.5**	**49.1**
Oracle	48.6	47.1	55.2	66.4	63.8	71.6	58.3	71.0	73.7

Table 5.1: mIoU on the respective target sets for 3D semantic segmentation in different cross-modal UDA scenarios. We report the result for each network stream (2D and 3D) as well as the ensembling result ('softmax avg'). PL = Pseudo-labeling.

terms of performance, the 'oracle' in Table 5.1 is trained supervisedly having access to labels on the target set and can be referred to as upper bound. We train the oracle on target set only, except the Day/Night oracle, where we used batches of 50%/50% Day/Night to prevent overfitting. Pseudo-labels (PL) are applied separately on each modality (2D pseudo-labels to train 2D, 3D pseudo-labels to train 3D).

In all three UDA scenarios xMUDA outperforms both normal and UDA baselines significantly, which proves the benefit of exchanging information between modalities. We observe that cross-modal learning and self-training with PL are complementary as the best score is always achieved combining both ('xMUDA'). This is expected as they represent different concepts: uni-modal self-training with pseudo-labels reinforces confident predictions while cross-modal learning enables knowledge sharing between modalities and can be seen as auxiliary task.

We observe that cross-modal learning consistently improves both modalities. Thus, even the strong modality can learn from the weaker one, thanks to decoupling of main and mimicking prediction.

Qualitative results are presented in Figure 5.7 where we only display the softmax average result to easily compare against the uni-modal UDA baseline (PL). As shown in Figure 5.7, xMUDA exhibits better segmentation across all three UDA scenarios which highlights the general benefit of cross-modal learning. Additional qualitative results are shown in Figure 5.8, depicting the individual 2D/3D outputs to illustrate their respective strengths and weaknesses. In Figure 5.8, we observe in 'A2D2 - SemanticKITTI' for the uni-modal 'UDA baseline (PL)', that while 2D tends to mistake the road as a building, 3D correctly predicts the road which is perhaps easier to classify based on the 3D geometry, but 3D can be fooled by classes with similar 3D shapes, e.g. it labels a building as truck. Note that in 'xMUDA', 2D and 3D are both much closer to the ground truth. Also in Figure 5.8 in the 'Day - Night' scenario, the 2D network of 'UDA baseline (PL)' performs poorly, because there is such a large large domain shift between day and night images. However, in 'xMUDA' *both* modalities perform well, even 2D

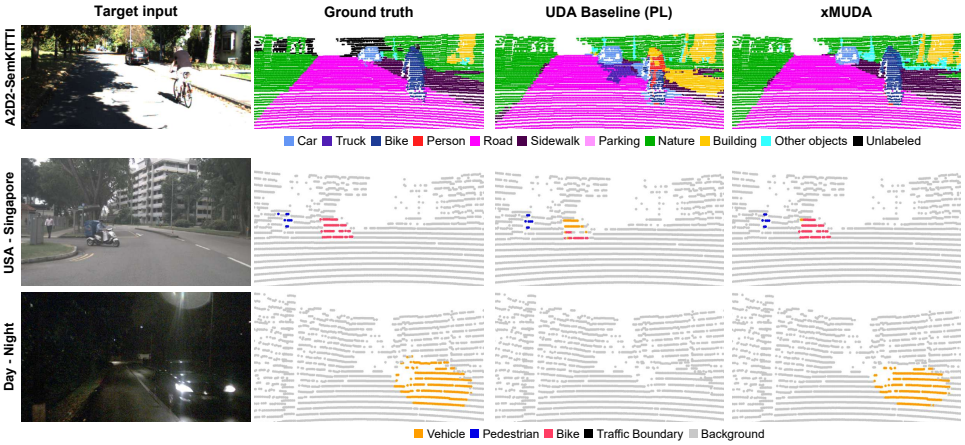


Figure 5.7: Qualitative results on the three proposed splits. We show the ensembling result obtained from averaging the softmax output of 2D and 3D on the UDA Baseline (PL) and xMUDA.
A2D2/SemanticKITTI: xMUDA helps to stabilize and refine segmentation performance when there are sensor changes (3x16 layer LiDAR with different angles to 64 layer LiDAR).
USA/Singapore: Delivery motorcycles with a storage box on the back are common in Singapore, but not in USA. The 3D shape might resemble a vehicle. However, 2D appearance information is leveraged in xMUDA to improve the recognition.
Day/Night: The visual appearance of a car at night with headlights turned on is very different than during day. The uni-modal UDA baseline is not able to learn this new appearance. However, if information between camera and robust-at-night LiDAR is exchanged in xMUDA, it is possible to detect the car correctly at night.

at night which seems to have learned how to recognize cars by their headlamps. We also provide a video of the 'A2D2 - SemanticKITTI' scenario at `http://tiny.cc/xmuda`.

5.4.4 Extension to Fusion

In the previous section we showed how each modality can be improved with xMUDA and consequently, the softmax average also increases. However, how can we obtain the best possible results by 2D and 3D *feature fusion*?

A common fusion architecture is late fusion where the features from different sources are concatenated (see Figure 5.9a). However, in this case there is only a single fused prediction head where it is not possible to apply cross-modal learning. Therefore, we propose xMUDA Fusion (see Figure 5.9b) where each modality has an *additional* uni-modal prediction output which is used to mimic the fusion prediction.

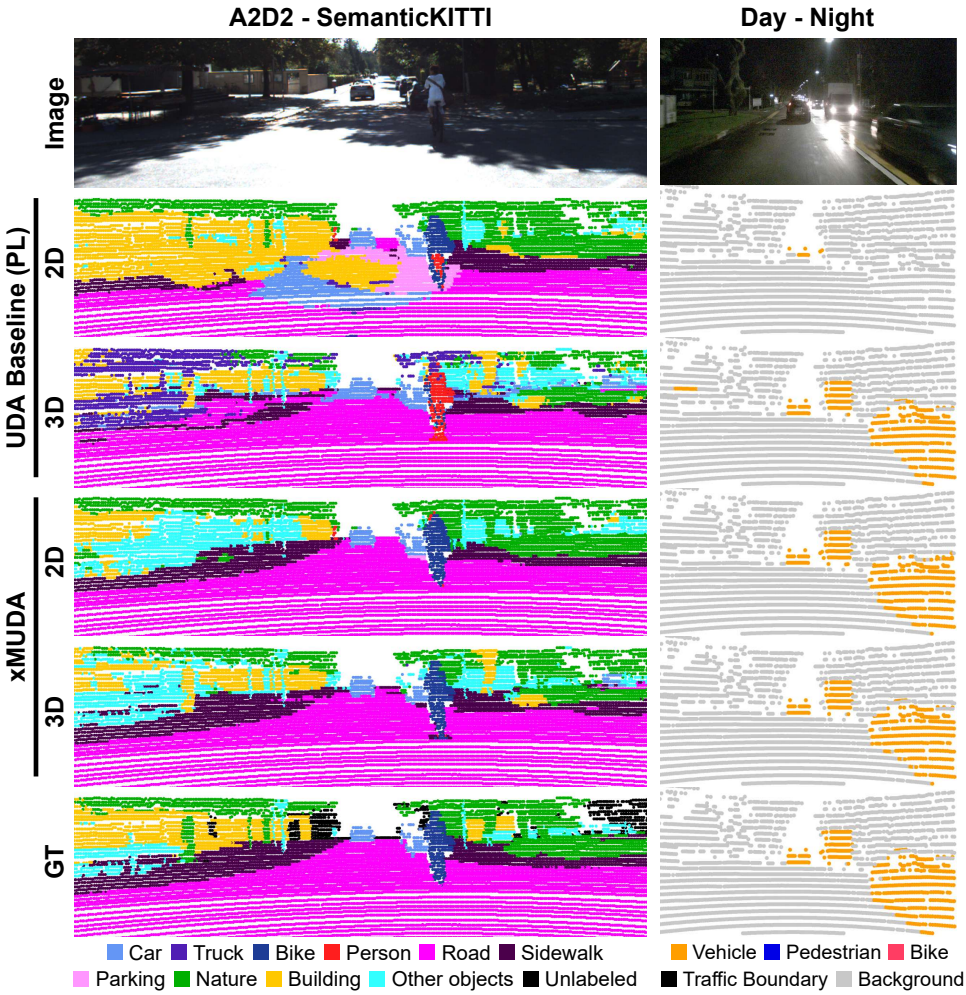


Figure 5.8: **Qualitative results on two UDA scenarios**. For UDA Baseline (PL) and xMUDA, we separately show the predictions of the 2D and 3D network stream.
A2D2/SemanticKITTI: For the uni-modal UDA baseline (PL), the 2D prediction lacks consistency on the road and 3D is unable to recognize the bike and the building on the left correctly. In xMUDA, both modalities can stabilize each other and obtain better performance on the bike, the road, the sidewalk and the building.
Day/Night: For the UDA Baseline, 2D can only partly recognize one car out of three while the 3D prediction is almost correct, with one false positive car on the left. With xMUDA, the 2D and 3D predictions are both correct.

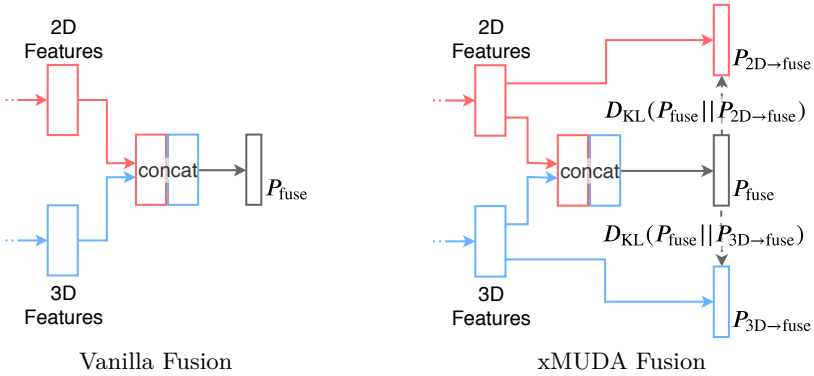
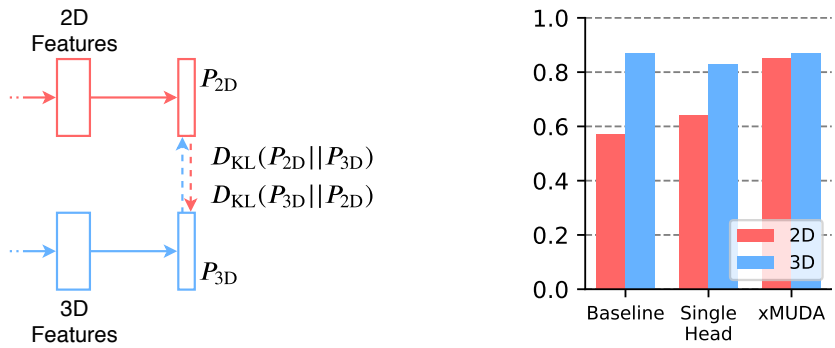


Figure 5.9: Architectures for late fusion. (a) In Vanilla Fusion the 2D and 3D features are concatenated, fed into a linear layer with ReLU to mix the features and followed by another linear layer and softmax to obtain a fused prediction $\boldsymbol{P}_{\text{fuse}}$. (b) In xMUDA Fusion, we add two uni-modal outputs $\boldsymbol{P}_{\text{2D}\to\text{fuse}}$ and $\boldsymbol{P}_{\text{3D}\to\text{fuse}}$ that are used to mimic the fusion output $\boldsymbol{P}_{\text{fuse}}$.

Vanilla Fusion (no UDA)	59.9
xMUDA Fusion w/o PL	61.9
Vanilla Fusion + PL	65.2
Distilled Vanilla Fusion	65.8
xMUDA Fusion	**66.6**
Oracle	72.2

Table 5.2: mIoU for fusion, USA/Singapore scenario.



Single head architecture

Mean probabilities (vehicle)

Figure 5.10: Single vs. Dual segmentation head. In (a), main and mimicry prediction are not uncoupled as in xMUDA of Figure 5.5. In (b), we compare mean predicted probabilities on points where the true label is vehicle. In the naive single head approach, 2D/3D probabilities are simply aligned, slightly decreasing the performance of the stronger 3D prediction, while the disentangled architecture of xMUDA with 2 segmentation heads uncouples the 2D improvement from the 3D result. Day/Night scenario.

In Table 5.2 we show results for different fusion approaches. 'xMUDA Fusion w/o PL' outperforms Vanilla Fusion thanks to cross-modal learning. We can improve over 'Vanilla Fusion + PL' with 'Distilled Vanilla Fusion' where we use the xMUDA model of the main experiments reported in Table 5.1 to generate pseudo-labels from the softmax average and train the Vanilla Fusion network. The best performance can be achieved with 'xMUDA Fusion', combining cross-modal learning and PL, analogously to the main experiments.

5.5 Ablation Studies

5.5.1 Segmentation Heads

In the following we justify our design choice of two segmentation heads per modality stream as opposed to a single one in a naive approach (see Figure 5.10a).

In the single head architecture the mimicking objective is directly applied between the two main predictions, which leads to an increase of probability in the weaker and a decrease in the stronger modality, as can be seen for the vehicle class in Figure 5.10b. There is shared information between 2D/3D, but also private information in each modality. An unwanted solution to reduce the cross-modal loss $\mathcal{L}_{\text{xM}}$ is that the networks discard private information, so that they both only use shared information making it easier

	Single head w/o PL			xMUDA w/o PL		
λ_t	2D	3D	softmax avg	2D	3D	softmax avg
0.001	52.8	47.9	60.5	51.4	47.9	59.1
0.01	52.4	48.8	58.4	52.4	49.1	60.4
0.1	43.9	40.8	46.5	**59.3**	**52.0**	**62.7**
1.0	24.7	23.1	21.5	54.6	49.1	57.0

Table 5.3: mIoU without PL of Single head and xMUDA (Dual head), while varying the weight of the cross-modal loss on target λ_t. USA/Singapore scenario.

to align their outputs. However, we can obviously achieve the best performance if the private information is also used. By separating the main from the mimicking prediction with dual segmentation heads, we can effectively decouple the two optimization objectives: The main head outputs the best possible prediction to optimize the segmentation loss, while the mimicking head can align with the other modality.

The experiments only include the first training step without PL as we want to benchmark the pure cross-modal learning. From results in Table 5.3, xMUDA has much better performance than the single head architecture and is also much more robust when it comes to choosing a good hyperparameter, specifically the weight of the cross-modal loss λ_t. As the loss weight λ_t becomes too large, single head performance decreases as the network resorts to the trivial solution of predicting the most frequent class to align 2D/3D outputs, while xMUDA performance is robust. Note that we fix λ_s optimally for each architecture, 0.1 for single head and 1.0 for xMUDA.

5.5.2 Cross-modal Learning on Source

In Equation 5.4, cross-modal loss $\mathcal{L}_{\text{xM}}$ is applied on source *and* target, although we already have supervised segmentation loss $\mathcal{L}_{\text{seg}}$ on source. We observe an improvement of 4.8 mIoU on 2D and 4.4 mIoU on 3D when adding $\mathcal{L}_{\text{xM}}$ on source as opposed to applying it on target *only*. This shows that it is important to train the mimicking head on source, stabilizing the predictions, which can be exploited during adaptation on target.

5.5.3 Cross-modal Learning for Oracle Training

We have shown that cross-modal learning is very effective for UDA. However, it can also be used in a purely supervised setting. When training the oracle with cross-modal loss $\mathcal{L}_{\text{xM}}$, we can improve over the baseline, see Table 5.4. We conjecture that $\mathcal{L}_{\text{xM}}$ is a beneficial auxiliary loss and can help to regularize training and prevent overfitting.

Method	2D	3D	softmax avg	Method	fusion
w/o $\mathcal{L}_{\text{xM}}$	65.8	63.2	71.1	Vanilla Fusion	71.0
with $\mathcal{L}_{\text{xM}}$	**66.4**	**63.8**	**71.6**	Fusion + $\mathcal{L}_{\text{xM}}$	**72.2**

Table 5.4: Cross-modal loss in supervised setting for oracle training. mIoU on USA/Singapore.

5.6 Conclusion

In this chapter we propose xMUDA, Cross-modal Unsupervised Domain Adaptation, where modalities learn from each other to improve performance on the target domain. For cross-modal learning we introduce mutual mimicking between the modalities, achieved through KL divergence. We design an architecture with separated main and mimicking head to disentangle the segmentation from the cross-modal learning objective. Experiments on 3D semantic segmentation on new UDA scenarios using 2D/3D datasets, show that xMUDA largely outperforms uni-modal UDA and is complementary to the pseudo-label strategy.

Different from feature fusion in preceding Chapters 3 and 4, we demonstrate here that performance can be improved on each modality without any feature fusion, i.e. just by exchanging information in output-space between the 2D and 3D branch. However, we also showcase how feature fusion can further boost performance, and that it benefits from the additional unsupervised mimicking loss.

We think that cross-modal learning could be useful in a wide variety of settings and tasks, not limited to UDA. For example, one could extend to other modalities, supervised learning or, as recently shown by Alwassel et al. (2019), to self-supervised learning. Specifically, Alwassel et al. (2019) leverage cross-modal pseudo-labels, obtained through feature clustering, to train feature encoders for the modalities of audio and video. In the case of representation learning for action recognition, they obtain better performance with their self-supervised method than with supervised pretraining on large-scale datasets.

Conclusion

French Summary of the Chapter "Conclusion"

Dans ce chaptire nous résumons nos contributions en conduite de bout en bout basé sur apprentissage par renforcement, la fusion de données pour l'amélioration d'apprentissage supervisé, ainsi que l'adaptation de domaine non-supervisé par l'apprentissage inter-modalités. De futures extensions de ces travaux sont aussi envisagées.

Chapter 6

Conclusion

6.1 Contributions

In this thesis, besides our work on end-to-end driving with reinforcement learning, we presented different approaches that aimed at jointly exploiting images and point clouds to improve performance in supervised learning as well as unsupervised learning. This was achieved through the design of architectures fusing features in 2D and 3D space, as well as the technique of mutual mimicking between modalities.

We applied state-of-the-art reinforcement learning techniques to end-to-end driving, designing driving specific reward functions and developing novel training strategies. To reduce the gap between simulation and real world driving, we trained in a simulator with realistic graphics and physics which are more demanding in computation. Therefore, to speed up training, we implemented distributed learning on multiple machines.

We proposed architectures to fuse images and point clouds in either 2D space or 3D space and carried out the tasks of depth completion and 2D/3D semantic segmentation. In both, we show that 2D-3D fusion leads to better performance than their uni-modal baselines and that dense image guidance is especially helpful for robustness against low-resolution point clouds.

We proposed a method to fuse multiple views into a global point cloud which consists of RGB feature computation in 2D, lifting features to 3D, fusing them and computing the final 3D semantic segmentation with a 3D network. This 2D-3D lifting strategy allows the aggregation of temporal data (multiple local views) in a natural global 3D point cloud representation.

We introduced the novel task of cross-modal Unsupervised Domain Adaptation (UDA) where one has access to multi-modal data on the source and target set. Furthermore, we provided new splits on existing datasets that represent interesting UDA scenarios for 3D semantic segmentation.

We proposed mutual mimicking between images and point clouds to address source-target domain shift, formalized as KL-divergence between output predictions of networks that take different modalities as input. We show the efficiency of this approach in terms of domain adaptation on our newly provided UDA scenarios. As our method operates differently on the data compared to existing uni-modal UDA techniques, it can be applied in a complementary fashion on top of these. We showcase above the state-of-the-art performance when combining cross-modal learning with the uni-modal

UDA technique of pseudo-label self-training.

We design an architecture that enables cross-modal learning by separating the 2D and 3D network streams and disentangles the mimicking from the main segmentation head. We showcase how this dual head strategy can also be applied on top of a fusion architecture, making it possible to address the domain shift separately in each modality.

6.2 Future Work

In the last work of this thesis we aimed at reducing the dependence on labels while focusing on the UDA task. I think that cross-modal learning between images and point clouds can potentially be extended to self-supervised learning of representations. Such "pretraining" on large amounts of unlabeled data can, in my opinion, lead to better performance than commonly practiced pretraining on ImageNet (Deng et al., 2009). The latter strategy has its limitations: for instance, the object-centered classification examples in ImageNet are different in layout, lighting conditions and content than images of outdoor driving or indoor scenes and there is no annotated point cloud dataset that likens the size of ImageNet. In contrast, self-supervised training on large-scale data could be performed on data that is closer to the target distribution. Indeed, *recording* data is usually simple and cheap compared to its annotation.

Existing works in self-supervision build on exploiting different constrains as learning signal, e.g. geometry constraints for monocular depth estimation using stereo camera pairs (Godard et al., 2019) or monocular video (Zhou et al., 2017). Another possible learning signal stems from co-registration of multiple modalities, e.g. for feature learning using audio-visual correspondence (Arandjelovic and Zisserman, 2017). Similarly, and somewhat related to our work presented in Chapter 5, Alwassel et al. (2019) generate cross-modal pseudo-labels from audio and video data to learn informative feature representations. In the same multi-modal spirit, interesting results have been shown in maximizing mutual information between modalities, i.e. depth, segmentation, luminance and color channel of an image (Tian et al., 2019). The underlying principles in the discussed works rely on either exploiting *geometric constraints* or *cross-modal correspondence* between image representations or image and audio. I think that in the multi-modal case of point clouds and images, we could combine both types of constraints, i.e. enforce geometric consistency as for instance between image-based depth estimation and point cloud, and at the same time semantic consistency of the learned features, e.g. through letting the network classify if a 2D and 3D point or an image and a point cloud correspond or not.

Our work on cross-modal learning for unsupervised domain adaptation deals with the case where absolutely no labels are available on the target set.

Another interesting research lead is weakly-supervised domain adaptation, where one considers to have access to a small number of annotations on the target set. In reality, this case is often encountered as one can usually afford to annotate a tiny portion of the data.

In UDA, one has access to data in the target domain and the network can be adapted to this kind of data. However, this still leaves the network totally unprepared for out-of-distribution events, for example unseen situations (the long tail of the data distribution) or sensor degradation (decalibration, soiling or failure). It is very important for perception systems to function or to detect such unforeseen adversary events when performance can be impacted. A promising way forward is to predict uncertainty of a network output as proposed by Kendall et al. (2017). Recent work from Tian et al. (2020) proposes to train three different expert networks (RGB-D, RGB, D) which outputs are combined based on different uncertainty measures. However, Tian et al. (2020) need to run three models in parallel which increases computation. It would be of high interest to learn a single robust fusion model.

Finally, we only looked at our data in a frame-wise fashion in this thesis. However, in reality data often comes as a sequence and fusing information across the temporal domain is a major avenue for performance improvement in scene understanding via the exploitation of causality. This could also help forecasting, because temporal data can provide clues about intentions of drivers, bikers and pedestrians, important to perform planning in autonomous driving.

Publications

This thesis led to the following publications:

- Perot, E., **Jaritz, M.**, Toromanoff, M., and de Charette, R.
  End-to-end driving in a realistic racing game with
  deep reinforcement learning.
  CVPR Workshop 2017.

- **Jaritz, M.**, de Charette, R., Toromanoff, M., Perot, E.,
  and Nashashibi, F.
  End-to-end race driving with deep reinforcement learning.
  ICRA 2018.

- **Jaritz, M.**, de Charette, R., Wirbel, E., Perrotton, X.,
  and Nashashibi, F.
  Sparse and dense data with CNNs: depth completion and
  semantic segmentation.
  3DV 2018.

- **Jaritz, M.**, Gu, J., and Su, H.
  Multi-view PointNet for 3D scene understanding.
  ICCV Workshop 2019.

- **Jaritz, M.**, Vu, T.-H., de Charette, R., Wirbel, E., and Pérez, P.
  xMUDA: Cross-Modal Unsupervised Domain Adaptation for 3D semantic segmentation.
  CVPR 2020.

Bibliography

Alwassel, H., Mahajan, D., Torresani, L., Ghanem, B., and Tran, D. (2019). Self-supervised learning by cross-modal audio-video clustering. *arXiv 2019*.

Arandjelovic, R. and Zisserman, A. (2017). Look, listen and learn. In *ICCV 2017*.

Armeni, I., Sax, S., Zamir, A. R., and Savarese, S. (2017). Joint 2d-3d-semantic data for indoor scene understanding. *arXiv 2017*.

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *TPAMI 2017*.

Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., and Gall, J. (2019). SemanticKITTI: A dataset for semantic scene understanding of LiDAR sequences. In *ICCV 2019*.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv 2016*.

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

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. (2015). Shapenet: An information-rich 3d model repository. *arXiv 2015*.

Chen, C., Seff, A., Kornhauser, A., and Xiao, J. (2015). Deepdriving: Learning affordance for direct perception in autonomous driving. In *ICCV 2015*.

Chen, D., Zhou, B., Koltun, V., and Krähenbühl, P. (2019a). Learning by cheating. *CoRL 2019*.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2018). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI 2018*.

Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. (2017a). Rethinking atrous convolution for semantic image segmentation. *arXiv 2017*.

Chen, X., Ma, H., Wan, J., Li, B., and Xia, T. (2017b). Multi-view 3d object detection network for autonomous driving. In *CVPR 2017*.

Chen, Y., Yang, B., Liang, M., and Urtasun, R. (2019b). Learning joint 2D-3D representations for depth completion. In *ICCV 2019*.

Cheng, X., Wang, P., Guan, C., and Yang, R. (2019). CSPN++: Learning context and resource aware convolutional spatial propagation networks for depth completion. *arXiv 2019*.

Cheng, X., Wang, P., and Yang, R. (2018). Depth estimation via affinity learned with convolutional spatial propagation network. In *ECCV 2018*.

Chiang, H.-Y., Lin, Y.-L., Liu, Y.-C., and Hsu, W. H. (2019). A unified point-based framework for 3D segmentation. In *3DV 2019*.

Choy, C., Gwak, J., and Savarese, S. (2019). 4D spatio temporal convnet: Minkowski convolutional neural networks. In *CVPR 2019*.

Çiçek, Ö., Abdulkadir, A., Lienkamp, S. S., Brox, T., and Ronneberger, O. (2016). 3d u-net: learning dense volumetric segmentation from sparse annotation. In *MICCAI 2016*.

Codevilla, F., Miiller, M., López, A., Koltun, V., and Dosovitskiy, A. (2018). End-to-end driving via conditional imitation learning. In *ICRA 2018*.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *CVPR 2016*.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018). Implicit quantile networks for distributional reinforcement learning. *ICML 2018*.

Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017). ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *CVPR 2017*.

Dai, A., Diller, C., and Nießner, M. (2019). Sg-nn: Sparse generative neural networks for self-supervised scene completion of rgb-d scans. *arXiv 2019*.

Dai, A. and Nießner, M. (2018). 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. In *ECCV 2018*.

Dai, A., Ritchie, D., Bokeloh, M., Reed, S., Sturm, J., and Nießner, M. (2018). Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans. In *CVPR 2018*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *CVPR 2009*.

DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. *arXiv 2017*.

Doersch, C., Gupta, A., and Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *ICCV 2015*.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). Carla: An open urban driving simulator. *arXiv 2017*.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *ICML 2016*.

Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *ICCV 1999*.

Eigen, D., Puhrsch, C., and Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. In *NeurIPS 2014*.

Eldesokey, A., Felsberg, M., and Khan, F. S. (2019). Confidence propagation through CNNs for guided sparse depth regression. *TPAMI 2019*.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., March, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *JMLR 2016*.

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *IJRR 2013*.

Geyer, J., Kassahun, Y., Mahmudi, M., Ricou, X., Durgesh, R., Chung, A. S., Hauswald, L., Pham, V. H., Mühlegg, M., Dorn, S., Fernandez, T., Jänicke, M., Mirashi, S., Savani, C., Sturm, M., Vorobiov, O., and Schuberth, P. (2019). A2D2: AEV autonomous driving dataset. http://www.a2d2.audi.

Godard, C., Mac Aodha, O., and Brostow, G. J. (2017). Unsupervised monocular depth estimation with left-right consistency. In *CVPR 2017*.

Godard, C., Mac Aodha, O., Firman, M., and Brostow, G. J. (2019). Digging into self-supervised monocular depth estimation. In *ICCV 2019*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *NIPS 2014*.

Graham, B., Engelcke, M., and van der Maaten, L. (2018). 3D semantic segmentation with submanifold sparse convolutional networks. In *CVPR 2018*.

Graham, B. and van der Maaten, L. (2017). Submanifold sparse convolutional networks. *arXiv 2017*.

Groh, F., Wieschollek, P., and Lensch, H. P. (2018). Flex-convolution (million-scale point-cloud learning beyond grid-worlds). In *ACCV 2018*.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *ICRA 2017*.

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous deep Q-learning with model-based acceleration. In *ICML 2016*.

Gupta, S., Girshick, R., Arbeláez, P., and Malik, J. (2014). Learning rich features from rgb-d images for object detection and segmentation. In *ECCV 2014*.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv 2018*.

Hackel, T., Savinov, N., Ladicky, L., Wegner, J. D., Schindler, K., and Pollefeys, M. (2017). SEMANTIC3D.NET: A new large-scale point cloud classification benchmark. In *ISPRS 2017*.

Hancock, M. W. and Wright, B. (2001). A policy on geometric design of highways and streets. *The American Association of State Highway and Transportation Officials*.

Hazirbas, C., Ma, L., Domokos, C., and Cremers, D. (2016). Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture. In *ACCV 2016*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR 2016*.

Hermosilla, P., Ritschel, T., Vázquez, P.-P., Vinacua, À., and Ropinski, T. (2018). Monte carlo convolution for learning on non-uniformly sampled point clouds. In *SIGGRAPH Asia 2018 Technical Papers*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *AAAI 2018*.

Hinton, G., Vinyals, O., and Dean, J. (2014). Distilling the knowledge in a neural network. In *NIPS Workshop 2014*.

Hoffman, J., Tzeng, E., Park, T., Zhu, J.-Y., Isola, P., Saenko, K., Efros, A., and Darrell, T. (2018). CyCADA: Cycle-consistent adversarial domain adaptation. In *ICML 2018*.

Hoffman, J., Wang, D., Yu, F., and Darrell, T. (2016). FCNs in the wild: Pixel-level adversarial and constraint-based adaptation. *arXiv 2016*.

Huang, J., Zhang, H., Yi, L., Funkhouser, T., Nießner, M., and Guibas, L. J. (2019). Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes. In *CVPR 2019*.

Jaritz, M., de Charette, R., Toromanoff, M., Perot, E., and Nashashibi, F. (2018a). End-to-end race driving with deep reinforcement learning. In *ICRA 2018*.

Jaritz, M., de Charette, R., Wirbel, E., Perrotton, X., and Nashashibi, F. (2018b). Sparse and dense data with CNNs: depth completion and semantic segmentation. In *3DV 2018*.

Jaritz, M., Gu, J., and Su, H. (2019). Multi-view PointNet for 3D scene understanding. In *ICCV Workshop 2019*.

Jaritz, M., Vu, T.-H., de Charette, R., Wirbel, E., and Pérez, P. (2020). xMUDA: Cross-Modal Unsupervised Domain Adaptation for 3D semantic segmentation. *CVPR 2020*.

Jiang, L., Zhao, H., Liu, S., Shen, X., Fu, C.-W., and Jia, J. (2019). Hierarchical point-edge interaction network for point cloud semantic segmentation. In *CVPR 2019*.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRL 2018*.

Kamnitsas, K., Ledig, C., Newcombe, V. F., Simpson, J. P., Kane, A. D., Menon, D. K., Rueckert, D., and Glocker, B. (2017). Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation. *MEDIA 2017*.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. (2016). Vizdoom: A doom-based ai research platform for visual reinforcement learning. *CIG 2016*.

Kendall, A., Badrinarayanan, V., and Cipolla, R. (2017). Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *BMVC 2017*.

Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A., and Shah, A. (2019). Learning to drive in a day. In *ICRA 2019*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS 2012*.

Ku, J., Harakeh, A., and Waslander, S. L. (2018). In defense of classical image processing: Fast depth completion on the cpu. *CRV 2018*.

Kuznietsov, Y., Stückler, J., and Leibe, B. (2017). Semi-supervised deep learning for monocular depth map prediction. In *CVPR 2017*.

Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., and Navab, N. (2016). Deeper depth prediction with fully convolutional residual networks. In *3DV 2016*.

Landrieu, L. and Simonovsky, M. (2018). Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR 2018*.

Lau, B. (2016). Using Keras and Deep Deterministic Policy Gradient to play TORCS.

Lee, D.-H. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *ICML Workshop 2013*.

Lee, K.-H., Ros, G., Li, J., and Gaidon, A. (2019). Spigan: Privileged adversarial learning from simulation. In *ICLR 2019*.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *JMLR 2016*.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *IJRR 2018*.

Li, Y., Bu, R., Sun, M., Wu, W., Di, X., and Chen, B. (2018). Pointcnn: Convolution on x-transformed points. In *NeurIPS 2018*.

Li, Y., Yuan, L., and Vasconcelos, N. (2019). Bidirectional learning for domain adaptation of semantic segmentation. In *CVPR 2019*.

Liang, M., Yang, B., Wang, S., and Urtasun, R. (2018). Deep continuous fusion for multi-sensor 3D object detection. In *ECCV 2018*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *ICLR 2016*.

Liu, M.-Y., Breuel, T., and Kautz, J. (2017). Unsupervised image-to-image translation networks. In *NIPS 2017*.

Long, J., Shelhamer, E., and Darrell, T. (2015a). Fully convolutional networks for semantic segmentation. In *CVPR 2015*.

Long, M., Cao, Y., Wang, J., and Jordan, M. (2015b). Learning transferable features with deep adaptation networks. In *ICML 2015*.

Luo, W., Li, Y., Urtasun, R., and Zemel, R. (2016). Understanding the effective receptive field in deep convolutional neural networks. In *NeurIPS 2016*.

Ma, F., Cavalheiro, G. V., and Karaman, S. (2019). Self-supervised sparse-to-dense: Self-supervised depth completion from lidar and monocular camera. In *ICRA 2019*.

Ma, F. and Karaman, S. (2018). Sparse-to-dense: Depth prediction from sparse depth samples and a single image. *ICRA 2018*.

Mairal, J., Sapiro, G., and Elad, M. (2008). Learning multiscale sparse representations for image and video restoration. *Multiscale Modeling & Simulation*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *ICML 2016*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. *NIPS Workshop 2013*.

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., et al. (2008). Junior: The stanford entry in the urban challenge. *Journal of Field Robotics 2008*.

Morerio, P., Cavazza, J., and Murino, V. (2018). Minimal-entropy correlation alignment for unsupervised deep domain adaptation. In *ICLR 2018*.

Murez, Z., Kolouri, S., Kriegman, D., Ramamoorthi, R., and Kim, K. (2018). Image to image translation for domain adaptation. In *CVPR 2018*.

Perot, E., Jaritz, M., Toromanoff, M., and de Charette, R. (2017). End-to-end driving in a realistic racing game with deep reinforcement learning. In *CVPR Workshop 2017*.

Pizzati, F., de Charette, R., Zaccaria, M., and Cerri, P. (2020). Domain bridge for unpaired image-to-image translation and unsupervised domain adaptation. *WACV 2020*.

Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In *NIPS 1989*.

Qi, C. R., Liu, W., Wu, C., Su, H., and Guibas, L. J. (2018). Frustum pointnets for 3d object detection from rgb-d data. In *CVPR 2018*.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. *CVPR 2017*.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS 2017*.

Qiu, J., Cui, Z., Zhang, Y., Zhang, X., Liu, S., Zeng, B., and Pollefeys, M. (2019). Deeplidar: Deep surface normal guided depth prediction for outdoor scene from sparse lidar data and single color image. In *CVPR 2019*.

Ren, M., Pokrovsky, A., Yang, B., and Urtasun, R. (2018). Sbnet: Sparse blocks network for fast inference. In *CVPR 2018*.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS 2015*.

Riegler, G., Ulusoy, A. O., and Geiger, A. (2017). Octnet: Learning deep 3d representations at high resolutions. In *CVPR 2017*.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *MICCAI 2015*.

Ros, G., Sellart, L., Materzynska, J., Vazquez, D., and Lopez, A. M. (2016). The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR 2016*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *ICML 2015*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv 2017*.

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2018). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics 2018*.

Shi, S., Wang, X., and Li, H. (2019). Pointrcnn: 3d object proposal generation and detection from point cloud. In *CVPR 2019*.

Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor segmentation and support inference from rgbd images. In *ECCV 2012*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *ICLR 2015*.

Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., and Funkhouser, T. (2017). Semantic scene completion from a single depth image. In *CVPR 2017*.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. *ICLR 2015 Workshop*.

Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.-H., and Kautz, J. (2018). SPLATNet: Sparse lattice networks for point cloud processing. In *CVPR 2018*.

Sun, B. and Saenko, K. (2016). Deep coral: Correlation alignment for deep domain adaptation. In *ECCV 2016*.

Sun, Z., Bebis, G., and Miller, R. (2006). On-road vehicle detection: A review. *TPAMI 2006*.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*. MIT press Cambridge.

Tang, J., Tian, F.-P., Feng, W., Li, J., and Tan, P. (2019). Learning guided convolutional network for depth completion. *arXiv 2019*.

Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., and Guibas, L. J. (2019). KPConv: Flexible and deformable convolution for point clouds. In *ICCV 2019*.

Tian, J., Cheung, W., Glaser, N., Liu, Y.-C., and Kira, Z. (2020). Uno: Uncertainty-aware noisy-or multimodal fusion for unanticipated input degradation. *ICRA 2020*.

Tian, Y., Krishnan, D., and Isola, P. (2019). Contrastive multiview coding. *arXiv 2019*.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *IROS 2012*.

Toromanoff, M., Wirbel, E., and Moutarde, F. (2020). End-to-end model-free reinforcement learning for urban driving using implicit affordances. *CVPR 2020*.

Tsai, Y.-H., Hung, W.-C., Schulter, S., Sohn, K., Yang, M.-H., and Chandraker, M. (2018). Learning to adapt structured output space for semantic segmentation. In *CVPR 2018*.

Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., and Darrell, T. (2014). Deep domain confusion: Maximizing for domain invariance. *arXiv 2014*.

Uhrig, J., Schneider, N., Schneider, L., Franke, U., Brox, T., and Geiger, A. (2017). Sparsity invariant CNNs. In *3DV 2017*.

Ummenhofer, B., Zhou, H., Uhrig, J., Mayer, N., Ilg, E., Dosovitskiy, A., and Brox, T. (2017). Demon: Depth and motion network for learning monocular stereo. In *CVPR 2017*.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C., et al. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics 2008*.

Valada, A., Mohan, R., and Burgard, W. (2019). Self-supervised model adaptation for multimodal semantic segmentation. *IJCV 2019*.

Valada, A., Vertens, J., Dhall, A., and Burgard, W. (2017). Adapnet: Adaptive semantic segmentation in adverse environmental conditions. In *ICRA 2017*.

Van Gansbeke, W., Neven, D., De Brabandere, B., and Van Gool, L. (2019). Sparse and noisy lidar completion with rgb guidance and uncertainty. *MVA 2019*.

Verma, N., Boyer, E., and Verbeek, J. (2018). Feastnet: Feature-steered graph convolutions for 3D shape analysis. In *CVPR 2018*.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D., and Silver, D. (2019). AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.

Vu, T.-H., Jain, H., Bucher, M., Cord, M., and Pérez, P. (2019a). Advent: Adversarial entropy minimization for domain adaptation in semantic segmentation. In *CVPR 2019*.

Vu, T.-H., Jain, H., Bucher, M., Cord, M., and Pérez, P. (2019b). DADA: Depth-aware domain adaptation in semantic segmentation. In *ICCV 2019*.

Wang, S., Suo, S., Ma, W.-C., Pokrovsky, A., and Urtasun, R. (2018). Deep parametric continuous convolutional neural networks. In *CVPR 2018*.

Wang, Y., Chao, W.-L., Garg, D., Hariharan, B., Campbell, M., and Weinberger, K. Q. (2019a). Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *CVPR 2019*.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019b). Dynamic graph cnn for learning on point clouds. *Transactions on Graphics 2019*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning 1992*.

Wu, B., Wan, A., Yue, X., and Keutzer, K. (2018a). Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3D LiDAR point cloud. In *ICRA 2018*.

Wu, B., Zhou, X., Zhao, S., Yue, X., and Keutzer, K. (2019a). Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. In *ICRA 2019*.

Wu, W., Qi, Z., and Fuxin, L. (2019b). PointConv: Deep convolutional networks on 3D point clouds. In *CVPR 2019*.

Wu, Z., Han, X., Lin, Y.-L., Gokhan Uzunbas, M., Goldstein, T., Nam Lim, S., and Davis, L. S. (2018b). DCAN: Dual channel-wise alignment networks for unsupervised scene adaptation. In *ECCV 2018*.

Wymann, B., Espié, E., Guionneau, C., Dimitrakakis, C., Coulom, R., and Sumner, A. (2000). Torcs, the open racing car simulator.

Xu, Y., Fan, T., Xu, M., Zeng, L., and Qiao, Y. (2018). Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *ECCV 2018*.

Xu, Y., Zhu, X., Shi, J., Zhang, G., Bao, H., and Li, H. (2019). Depth completion from sparse lidar data with depth-normal constraints. In *ICCV 2019*.

Yang, Y., Wong, A., and Soatto, S. (2019). Dense depth posterior (ddp) from single image and sparse range. In *CVPR 2019*.

You, Y., Wang, Y., Chao, W.-L., Garg, D., Pleiss, G., Hariharan, B., Campbell, M., and Weinberger, K. Q. (2020). Pseudo-LiDAR++: Accurate depth for 3D object detection in autonomous driving. *ICLR 2020*.

Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., and Huang, T. S. (2018). Generative image inpainting with contextual attention. In *CVPR*.

Zhang, Y., David, P., and Gong, B. (2017). Curriculum domain adaptation for semantic segmentation of urban scenes. In *ICCV 2017*.

Zhang, Y. and Funkhouser, T. (2018). Deep depth completion of a single rgb-d image. *CVPR 2018*.

Zhang, Y., Xiang, T., Hospedales, T. M., and Lu, H. (2018). Deep mutual learning. In *CVPR 2018*.

Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2017). Pyramid scene parsing network. In *CVPR 2017*.

Zhou, T., Brown, M., Snavely, N., and Lowe, D. G. (2017). Unsupervised learning of depth and ego-motion from video. In *CVPR 2017*.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. *CVPR 2018*.

Zou, Y., Yu, Z., Liu, X., Kumar, B. V., and Wang, J. (2019). Confidence regularized self-training. In *ICCV 2019*.

Appendix A

Fusing sparse depth and dense RGB for depth completion

In the following we will detail the network architecture used in Chapter 3.

A.1 Architecture details

We use the Tensorflow library. For the encoder, we use the original implementation of NASNet with the slim package[1]. We choose the mobile version (4@1056) of NASNet with the ImageNet stem and remove batch norm for the sparse depth branch. For the skip connections between encoder and decoder, we always took the highest level features at each resolution stage of the encoder, which are numbered in the original implementation as (from small to large resolution) ['Cell_7', 'Cell_3', 'Stem_1_Reduction_Cell_0', 'Stem_0_Strided_Conv'].

Our custom decoder is detailed in Table A.1. Note that for semantic segmentation the number of output channels for the last convolution is set to the number of classes.

[1] https://github.com/tensorflow/models/tree/master/research/slim

Operation	Input channels	Output channels	Up-/Down-sampling
Encoder (NASNet 4@1056), duplicated for each modality (sD and RGB)			
conv (s=2, k=3) *(Stem_0_Strided_Conv)*	1 (sD) or 3 (RGB)	32	1/2
reduction cell *(Stem_1_Reduction_Cell_0)*	32	44	1/2
reduction cell	44	88	1/2
normal cell *(Cell_0)*	88	264	
normal cell *(Cell_1)*	264	264	
normal cell *(Cell_2)*	264	264	
normal cell *(Cell_3)*	264	264	
reduction cell	264	352	1/2
normal cell *(Cell_4)*	352	528	
normal cell *(Cell_5)*	528	528	
normal cell *(Cell_6)*	528	528	
normal cell *(Cell_7)*	528	528	
reduction cell	528	704	1/2
normal cell *(Cell_8)*	704	1056	
normal cell *(Cell_9)*	1056	1056	
normal cell *(Cell_10)*	1056	1056	
normal cell *(Cell_11)*	1056	1056	
Custom Decoder			
concat features *from RGB + sD branch Cell_11*	2x1056	2112	
conv (s=1, k=3)	2112	1056	
transposed conv (s=2, k=3)	1056	528	2x
concat features *from RGB + sD branch Cell_7*	2x528	1056	
conv (s=1, k=3)	1056	528	
transposed conv (s=2, k=3)	528	264	2x
concat features *from RGB + sD branch Cell_3*	2x264	528	
conv (s=1, k=3)	528	264	
transposed conv (s=2, k=3)	264	132	2x
concat features *from RGB + sD branch Stem_1_Reduction_Cell_0*	2x132	264	
conv (s=1, k=3)	264	132	
transposed conv (s=2, k=3)	132	66	2x
concat features *from RGB + sD branch Stem_0_Strided_Conv*	66	132	
conv (s=1, k=3)	132	66	
transposed conv (s=2, k=3)	66	33	2x
conv (s=1, k=1) [depth regression]	33	1	

Table A.1: Details of our encoder-decoder architecture. The encoder is NASNet mobile 4@1056, where 4 refers to the number of 'normal cell' repetitions and 1056 to the output channel size of the last layer. We report the layer names of the official NASNet implementation in Tensorflow slim, e.g. *Cell_0*. In our custom decoder, we use ReLU activations after each convolution and transposed convolution.

Appendix B

Multi-view PointNet for 3D scene understanding

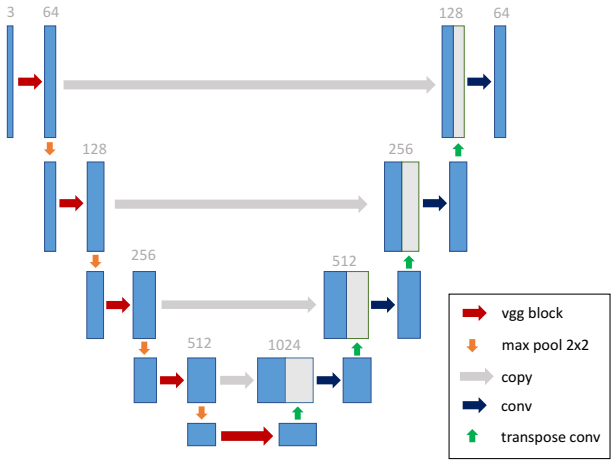


Figure B.1: The architecture of the 2D encoder-decoder network.

To supplement Chapter 4, we here show a detailed figure of our 2D network architecture and provide additional results for our submission to the ScanNet benchmark.

B.1 2D Encoder Decoder Architecture

Figure B.1 illustrates the architecture of the 2D encoder-decoder network inspired by U-Net (Ronneberger et al., 2015). We use VGG16 as encoder and initialize with ImageNet pre-trained weights. In the decoder, convolution is used to fuse concatenated features from skip connections, and transposed convolution for upsampling.

B.2 Additional Ablation Studies

For the ScanNetV2 3D semantic label benchmark, we employ MVPNet with 5 views and use ResNet34 as the 2D backbone. The numbers of centroids

Method	mIoU
MVPNet(VGG19)	66.6
MVPNet(ResNet34)	67.3
MVPNet(ResNet34) + class weights	68.0
MVPNet(ResNet34) + ensemble	68.3

Table B.1: The variants of MVPNet. The results are reported on the validation set of ScanNetV2.

are 2048, 512, 128, 64 respectively.

Table B.1 shows the comparison among several variants of the submission version. The stronger 2D backbone (ResNet34) improves the mIoU by 0.7 against the weaker 2D backbone (VGG19). Moreover, we also experiment with training MVPNet with class weights, which boosts the mIoU (+0.7) as the evaluation metric favors more balanced predictions. To achieve the best performance (68.3), we ensemble 4 models of MVPNet with ResNet34.

Appendix C

2D-3D Cross-Modal Unsupervised Domain Adaptation

In the following, we give details for the dataset splits of our Unsupervised Domain Adaptation (UDA) scenarios proposed in Chapter 5.

C.1 Dataset Splits

C.1.1 nuScenes

The nuScenes dataset (Caesar et al., 2019) consists of 1000 driving scenes, each of 20 seconds, which corresponds to 40k annotated keyframes taken at 2Hz. The scenes are split into train (28,130 keyframes), validation (6,019 keyframes) and hidden test set. The point-wise 3D semantic labels are obtained from 3D boxes like in (Wu et al., 2018a). We propose the following splits destined for domain adaptation with the respective source/target domains: Day/Night and USA/Singapore. Note that the USA data was captured in Boston. We use the official validation set as test set and divide the training set into train/val for the target set (see Table C.1 for the number of frames in each split). As the number of object instances in the target split can be very small (e.g. for night), we merge the objects into 5 categories: **vehicle** (car, truck, bus, trailer, construction vehicle), **pedestrian**, **bike** (motorcycle, bicycle), **traffic boundary** (traffic cone, barrier) and **background**.

	source		target		
Split	train	test	train	val	test
Day - Night	24,745	5,417	2,779	606	602
USA - Singapore	15,695	3,090	9,665	2,770	2,929
A2D2 - SemanticKITTI	27,695	942	18,029	1,101	4,071

Table C.1: Number of frames for the 3 splits. Day-Night and USA-Singapore are splits on nuScenes.

C.1.2 A2D2 and SemanticKITTI

The A2D2 dataset (Geyer et al., 2019) features 20 drives, which corresponds to 28,637 frames. The point cloud comes from three 16-layer front LiDARs (left, center, right) where the left and right front LiDARs are inclined. The semantic labeling was carried out in the 2D image for 38 classes and we compute the 3D labels by projection of the point cloud into the labeled image. We keep scene 20180807_145028 as test set and use the rest for training.

The SemanticKITTI dataset (Behley et al., 2019) provides 3D point cloud labels for the Odometry dataset of Kitti (Geiger et al., 2013) which features large angle front camera and a 64-layer LiDAR. The annotation of the 28 classes has been carried out directly in 3D.

We use the scenes $\{0, 1, 2, 3, 4, 5, 6, 9, 10\}$ as train set, 7 as validation and 8 as test set. We select 10 shared classes between the 2 datasets by merging or ignoring them (see Table C.2). The 10 final classes are **car**, **truck**, **bike**, **person**, **road**, **parking**, **sidewalk**, **building**, **nature**, **other-objects**.

A2D2 class	mapped class	SemanticKITTI class	mapped class
Car 1	car	unlabeled	ignore
Car 2	car	outlier	ignore
Car 3	car	car	car
Car 4	car	bicycle	bike
Bicycle 1	bike	bus	ignore
Bicycle 2	bike	motorcycle	bike
Bicycle 3	bike	on-rails	ignore
Bicycle 4	bike	truck	truck
Pedestrian 1	person	other-vehicle	ignore
Pedestrian 2	person	person	person
Pedestrian 3	person	bicyclist	bike
Truck 1	truck	motorcyclist	bike
Truck 2	truck	road	road
Truck 3	truck	parking	parking
Small vehicles 1	bike	sidewalk	sidewalk
Small vehicles 2	bike	other-ground	ignore
Small vehicles 3	bike	building	building
Traffic signal 1	other-objects	fence	other-objects
Traffic signal 2	other-objects	other-structure	ignore
Traffic signal 3	other-objects	lane-marking	road
Traffic sign 1	other-objects	vegetation	nature
Traffic sign 2	other-objects	trunk	nature
Traffic sign 3	other-objects	terrain	nature
Utility vehicle 1	ignore	pole	other-objects
Utility vehicle 2	ignore	traffic-sign	other-objects
Sidebars	other-objects	other-object	other-objects
Speed bumper	other-objects	moving-car	car
Curbstone	sidewalk	moving-bicyclist	bike
Solid line	road	moving-person	person
Irrelevant signs	other-objects	moving-motorcyclist	bike
Road blocks	other-objects	moving-on-rails	ignore
Tractor	ignore	moving-bus	ignore
Non-drivable street	ignore	moving-truck	truck
Zebra crossing	road	moving-other-vehicle	ignore
Obstacles / trash	other-objects		
Poles	other-objects		
RD restricted area	road		
Animals	other-objects		
Grid structure	other-objects		
Signal corpus	other-objects		
Drivable cobbleston	road		
Electronic traffic	other-objects		
Slow drive area	road		
Nature object	nature		
Parking area	parking		
Sidewalk	sidewalk		
Ego car	car		
Painted driv. instr.	road		
Traffic guide obj.	other-objects		
Dashed line	road		
RD normal street	road		
Sky	ignore		
Buildings	building		
Blurred area	ignore		
Rain dirt	ignore		

Table C.2: Class mapping for A2D2 - SemanticKITTI UDA scenario.

RÉSUMÉ

Dans cette thèse, nous abordons les défis de la rareté des annotations et la fusion de données hétérogènes tels que les nuages de points 3D et images 2D.

D'abord, nous adoptons une stratégie de *conduite de bout en bout* où un réseau de neurones est entraîné pour directement traduire l'entrée capteur (image caméra) en contrôles-commandes, ce qui rend cette approche indépendante des annotations dans le domaine visuel. Nous utilisons l'apprentissage par renforcement profond où l'algorithme apprend de la récompense, obtenue par interaction avec un simulateur réaliste. Nous proposons de nouvelles stratégies d'entraînement et fonctions de récompense pour une meilleure conduite et une convergence plus rapide. Cependant, le temps d'apprentissage reste élevé. C'est pourquoi nous nous concentrons sur la perception dans le reste de cette thèse pour étudier la fusion de nuage de points et d'images.

Nous proposons deux méthodes différentes pour la *fusion 2D-3D*. Premièrement, nous projetons des nuages de points LiDAR 3D dans l'espace image 2D, résultant en des cartes de profondeur éparses. Nous proposons une nouvelle architecture encodeur-décodeur qui fusionne les informations de l'image et la profondeur pour la tâche de complétion de carte de profondeur, améliorant ainsi la résolution du nuage de points projeté dans l'espace image. Deuxièmement, nous fusionnons directement dans l'espace 3D pour éviter la perte d'informations dû à la projection. Pour cela, nous calculons les caractéristiques d'image issues de plusieurs vues avec un CNN 2D, puis nous les projetons dans un nuage de points 3D global pour les fusionner avec l'information 3D. Par la suite, ce nuage de point enrichi sert d'entrée à un réseau "point-based" dont la tâche est l'inférence de la sémantique 3D par point.

Sur la base de ce travail, nous introduisons la nouvelle tâche *d'adaptation de domaine non supervisée inter-modalités* où on a accès à des données multi-capteurs dans une base de données source annotée et une base cible non annotée. Nous proposons une méthode d'apprentissage inter-modalités 2D-3D via une imitation mutuelle entre les réseaux d'images et de nuages de points pour résoudre l'écart de domaine source-cible. Nous montrons en outre que notre méthode est complémentaire à la technique unimodale existante dite de pseudo-labeling.

MOTS CLÉS

segmentation sémantique 3D, fusion 2D-3D, adaptation de domaine non supervisée, complétion de profondeur, nuage de points, conduite de bout en bout

ABSTRACT

In this thesis, we address the challenges of label scarcity and fusion of heterogeneous 3D point clouds and 2D images.

We adopt the strategy of *end-to-end race driving* where a neural network is trained to directly map sensor input (camera image) to control output, which makes this strategy independent from annotations in the visual domain. We employ deep reinforcement learning where the algorithm learns from reward by interaction with a realistic simulator. We propose new training strategies and reward functions for better driving and faster convergence. However, training time is still very long which is why we focus on perception to study point cloud and image fusion in the remainder of this thesis.

We propose two different methods for *2D-3D fusion*. First, we project 3D LiDAR point clouds into 2D image space, resulting in sparse depth maps. We propose a novel encoder-decoder architecture to fuse dense RGB and sparse depth for the task of depth completion that enhances point cloud resolution to image level. Second, we fuse directly in 3D space to prevent information loss through projection. Therefore, we compute image features with a 2D CNN of multiple views and then lift them all to a global 3D point cloud for fusion, followed by a point-based network to predict 3D semantic labels.

Building on this work, we introduce the more difficult novel task of *cross-modal unsupervised domain adaptation*, where one is provided with multi-modal data in a labeled source and an unlabeled target dataset. We propose to perform 2D-3D cross-modal learning via mutual mimicking between image and point cloud networks to address the source-target domain shift. We further showcase that our method is complementary to the existing uni-modal technique of pseudo-labeling.

KEYWORDS

3D semantic segmentation, 2D-3D fusion, unsupervised domain adaptation, depth completion, point cloud, end-to-end driving