



RHEINISCHE  
FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

BACHELOR THESIS

**Road-Object Segmentation from Sequences of  
Organized 3D-LIDAR Point Clouds**

*Author:*  
Moritz MÜLLER

*First Examiner:*  
Prof. Dr. Sven BEHNKE

*Second Examiner:*  
Priv. Doz. Dr. Volker STEINHAGE

*Supervisor:*  
Jan QUENZEL

*Department VI:*  
Autonomous Intelligent Systems

Date: January 19, 2019



# Sworn statement according to BAPO 2011 §17 Abs. 7

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

---

Place, Date

---

Signature





# Abstract

As autonomous intelligent systems play a more significant role in our society, 3D-LIDAR point segmentation approaches for recognizing frequent objects in urban environments have been noticeably improved over the last years. Recent works provide Convolutional Neural Network (CNN) architectures which demonstrate remarkable accuracy and runtime to address real-time applications such as autonomous driving. However, most of these approaches do not exploit the organized structure of LIDAR data, and none of them consider the full field of view (FoV) or temporal information to predict an object. One of the few exceptions is *SqueezeSeg*, which utilizes the organized structure to handle the high sparsity of 3D LIDAR point clouds. Inspired by rigid motion and optical flow algorithms, this work aims at embedding information from scan sequences into *SqueezeSeg* and extending it to the full FoV to examine the impact on segmentation results.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Foundation</b>	<b>3</b>
2.1	Operating Principle of LIDAR Scanner . . . . .	3
2.2	LIDAR Scanner Advantages and Limitations . . . . .	4
2.3	Calculating Organized Structured Point Cloud . . . . .	5
2.4	Description of Image and LIDAR Motion . . . . .	8
<b>3</b>	<b>Related Work</b>	<b>11</b>
3.1	LIDAR Segmentation . . . . .	11
3.2	Leveraging Temporal Information . . . . .	12
3.3	SqueezeSeg . . . . .	13
3.3.1	Custom Convolutional Modules . . . . .	13
3.3.2	CNN Architecture . . . . .	14
3.3.3	Limitations of SqueezeSeg . . . . .	16
<b>4</b>	<b>Overview of Methodology</b>	<b>17</b>
4.1	Datasets . . . . .	17
4.1.1	The KITTI Vision Benchmark . . . . .	18
4.1.2	A Dataset for Semantic Segmentation of Point Cloud Sequences . . . . .	21
4.1.3	Dataset Comparison . . . . .	22
4.2	Ground Truth Pose Retrieval from GPS and IMU Measurements . . . . .	23
4.3	Experimental Approach . . . . .	25
4.3.1	Extension 1: Input Robustness through Feature Concatenation . . . . .	26
4.3.2	Extension 2: Augmenting Present LIDAR Input Using Ego-motion Estimates . . . . .	28
4.3.3	Extension 3: Combining Successive Activations . . . . .	31
4.4	Training and Parameter Optimization . . . . .	35
4.4.1	Loss Function . . . . .	35
4.4.2	Momentum Optimization . . . . .	36
4.4.3	Training Environment and Evaluation Metrics . . . . .	38
4.4.4	Selection of the Hyperparameters . . . . .	38

*Contents*

4.5	Point Cloud Normalization . . . . .	40
<b>5</b>	<b>Evaluation</b>	<b>41</b>
5.1	Parameter and Runtime Analysis . . . . .	41
5.2	Experimental Metric Results . . . . .	43
5.3	Experimental Results Anaysis . . . . .	47
5.3.1	Extension 1.1: Naïve Concatenation . . . . .	47
5.3.2	Extension 1.2: Warping Concatenation . . . . .	47
5.3.3	Extension 2: Augmenting Current LIDAR Input Using Ego- motion Estimates . . . . .	48
5.3.4	Extension 3: Activation Combination and Activation Warping	48
<b>6</b>	<b>Conclusion</b>	<b>51</b>

# List of Figures

2.1	Structure and operating principle of a LIDAR scanner. . . . .	4
2.2	Process of point cloud transformation. . . . .	6
3.1	Structure of the <i>FireModule</i> from <i>SqueezeNet</i> . . . . .	14
3.2	Network architecture of <i>SqueezeSeg</i> . . . . .	14
3.3	Structure of a <i>FireModule</i> (left) and a <i>FireDeconv</i> (right). . . . .	15
4.1	Total number of object detections. . . . .	18
4.2	Total number of object instances. . . . .	18
4.3	Ground truth assignment by oriented bounding boxes. . . . .	20
4.4	Consecutive point clouds do not match well (left). The correct transformation reduces the discrepancy between the objects (right). . . . .	25
4.5	Feature channel concatenation of prior $OSP_{t-1}$ and current $OSP_t$ . . . . .	27
4.6	<b>Image warping:</b> $OSP_t^{t-1}$ includes the 2D displacement between time step $t$ and $t - 1$ in contrast to $OSP_{t-1}$ . . . . .	28
4.7	Bird’s eye view after ARV with augmented points (purple). . . . .	30
4.8	<b>Extension 3:</b> Combining successive activations through <i>Fuse Activation modules</i> . . . . .	32
4.9	<b>Fuse Activation Module:</b> Fuses prior and present activations. . . . .	33
5.1	<b>Evaluated dataset comparison:</b> Magnitude of applied parameters for each extension. . . . .	42
5.2	<b>Evaluated dataset comparison:</b> Detection runtime for each extension. . . . .	43
5.3	<b>Segmentation results:</b> 360° FoV Dataset. . . . .	45
5.4	<b>Segmentation results:</b> 360° FoV Dataset using 16 vertical beams. . . . .	46



# List of Tables

4.1	<b>Label reassignment:</b> Raw label of the 360° FoV dataset is assigned to our primary classes. . . . .	21
4.2	LIDAR datasets split in training, validation and test set. . . . .	22
4.3	Significant dataset differences at the pixel label distribution. . . . .	22
4.4	The ARV module exploits prior point clouds to complete empty entries. . . . .	30
4.5	Overview of utilized hyperparameters. . . . .	39
4.6	<b>Training comparison:</b> Randomized batches exhibit significant better IoU than ordered batches. . . . .	39
5.1	<b>Segmentation results:</b> KITTI's 81° FoV Dataset. . . . .	44
5.2	<b>Segmentation results:</b> 360° FoV Dataset. . . . .	44
5.3	<b>Segmentation results:</b> 360° FoV Dataset using 16 vertical beams. . . . .	46





# 1 Introduction

Light detection and ranging (LIDAR) scanners constitute an indispensable part of modern autonomous cars and micro aerial vehicles (MAV). This sensor technology allows to perceive the environment in 3D which covers a sufficient FoV in real-time. LIDAR is independent of ambient illumination and provides high-accuracy depth measurements in contrast to cameras. Hence, there is a considerable benefit to detect ordinary urban objects using efficient 3D LIDAR segmentation. Autonomous vehicles utilize this point-wise information, inter alia, for accurate object identification, obstacle avoidance, and autonomous navigation to interact safely and reliably in their environment.

The research field addressing LIDAR segmentation demonstrated significant enhancements during the last years and showed various strategies [7, 23, 29, 30]. A considerable number of recent work proposed the usage of CNNs which also demonstrate state-of-the-art segmentation performance in similar fields [2]. However, these methods have some limitations:

One can observe that sparse LIDAR point clouds are usually insufficient since some surfaces prevent the reflection of emitted beams. The major drawback of recent LIDAR-based CNN approaches is that they are limiting to a naïve single input mechanism, and none of them considers the temporal or prior information between a sequential frame sequence to improve the robustness and consistency of the outcome.

Another considerable weakness of recent work is that most of them using 3D LIDAR input representations such as point clouds which entail high input sparsity and complexity as well as excessively high computational effort. As a result, most of them do not comply with the requirements of limited hardware regarding autonomous vehicles.

To the best of our knowledge, there does not exist a survey on how to embed prior information into a LIDAR-based segmentation CNN that considers the FoV and solely relies on range data. This thesis aims to close this gap by evaluating different strategies:

First, we base our work on *SqueezeSeg* [30] which yields a superior efficiency and exploits the organized image-like structure of the point cloud. Then, we developed three main experiments to examine the impact of different prior representations

## 1 Introduction

on segmentation results regarding chronological frame sequences. In the first and second experiments, we use prior LIDAR scans to augment the current input representation of *SqueezeSeg*. Furthermore, we utilize the 6D rigid motion provided by the GPS. This temporal information allows us to transform the pose of previous point clouds into the next frame. The last experiment aims to study the influence of combining previous and present internal activation maps on *SqueezeSeg*. Moreover, this experiment considers the changed feature channel arrangement by warping the previous activation map into the representation of the activation frame.

Building upon these results, we hypothesize that the combination of rigid motion, as well as prior frames, demonstrate an added value to the segmentation process and hence we expect better segmentation results. We evaluated all experiments on two separate LIDAR datasets [11, 3] and show the results as well as their corresponding interpretation at the end of this thesis. The point clouds stem from a LIDAR scanner attached to a car with a vertical resolution of 64. However, LIDAR sensors suitable for MAVs have only 16 vertical beams. It is also essential to evaluate the performance for all extensions on less environment information and hence the last chapter additionally provides results based on point clouds with reduced vertical resolution.

## 2 Foundation

This chapter describes the operating principle of an ordinary LIDAR scanner and outlines their benefits as well as limitations. As the 3D point clouds tend to be sparse, we afterward present a technique to acquire a more compact representation. Finally, the last section highlights some techniques to describe the motion between an adjacent pair of images or 3D point clouds, respectively.

### 2.1 Operating Principle of LIDAR Scanner

LIDAR is nowadays an essential technology for measuring distances precisely and reliably. In general, LIDAR devices emit laser pulses with a specific wavelength towards the environment. The receiving object absorbs a specific fraction of transmitted energy depending on their material and surface structure before reflecting the laser beam. If the reflection heads back towards the sensor, a photoelectric cell made of silicon or gallium arsenide measures the received beam. This measurement represents the reflection strength  $r$  of the laser beam and varies by distance as well as inclination angle and material. In particular, it underlines that the reflectance measurement is more relative than quantifiable. Considering the constant speed of light  $v_{light} \approx 300,000 \text{ km/s}$  as well as the time of flight  $t_{tof}$  of a returning light photon, we can determine the range between sensor and explored object as follows:

$$d = \frac{1}{2} \cdot v_{light} \cdot t_{tof}. \quad (2.1.1)$$

LIDAR scanners utilize oscillating mirrors to transmit laser pulses rapidly on different elevation  $\phi$  and azimuth  $\theta$  level as illustrated in Figure 2.1(a). Moreover, we can summarize the range  $d$  together with the corresponding  $\theta$  and  $\phi$  angle as a point  $\mathbf{p}_{pol} = (\phi, \theta, d) \in \mathbb{R}^3$  located in 3D polar coordinate system. One can observe that the total number of reflected points differ by each scan because it depends on the reflection behavior of each surrounding area. Figure 2.1(b) demonstrates a visualized 3D point cloud. The resulting point cloud density depends on the vertical and horizontal angular resolutions and varies by each scanner. Besides, LIDAR scanner can explore the  $360^\circ$  FoV at a high frame rate of up to 20Hz

## 2 Foundation

which makes them suitable for applications like obstacle detection and navigation for autonomous driving.

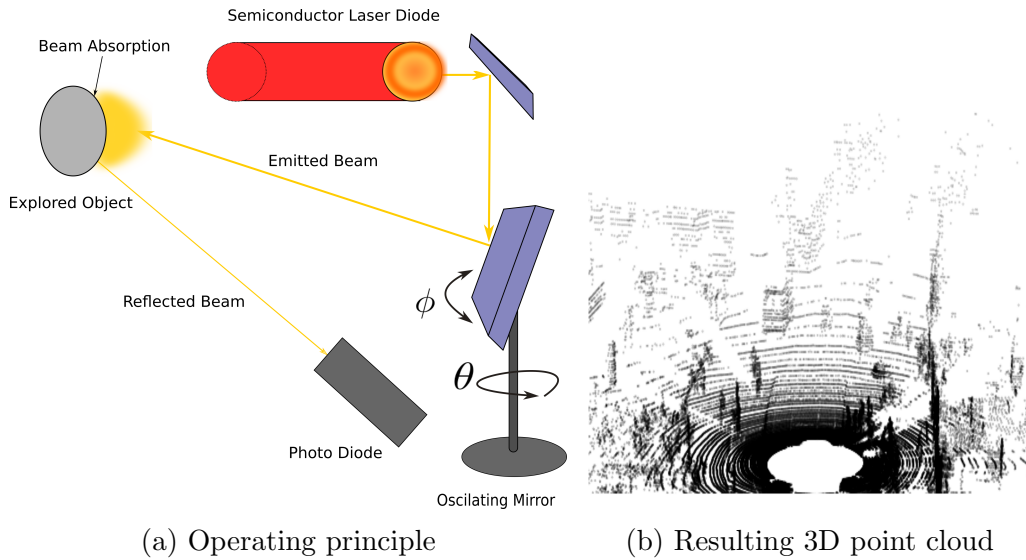


Figure 2.1: Structure and operating principle of a LIDAR scanner.

Other applications, e.g. Airborne LIDAR scanning, utilize short wavelengths in order to transmit beams with high energy and hence can detect distances with far greater range. It should be noted that with a shorter wavelength of the LIDAR beam the risk to cause damage to human eyes increases. However, the mounted LIDAR scanner on autonomous vehicles such as the MAV and self-driving cars commonly emit wavelengths between 900 and 1100  $nm$  to cover the full environment up to a range of 100 meters. These wavelengths are imperceptible for human eyes and also constitute a low-level risk of injury.

## 2.2 LIDAR Scanner Advantages and Limitations

One key advantage of laser scanners constitutes their ability to collect 360 degree 3D maps with outstanding distance accuracy in real-time. Another benefit is the fact that all measurements are independent of ambient illumination. In poorly illuminated environments like total darkness or sunny weather, laser scanners provide more valuable information about the surrounding area in contrast to photogrammetric sensors. However, this system also has some serious drawbacks. Engineers should take into account at any rate, that the reflectance measurement  $r$  is an unreliable property. The distribution of 3D LIDAR point clouds is generally sparse as well as irregular. It also requires an appropriate method for analyzing such huge

dataset efficiently. To overcome the high point sparsity issue, we will later compute a dense, organized, structured grid representation as described in Section 2.3. A photogrammetric setup of several cameras which includes the same FoV deals with geometrical distortions and higher instability caused by the sensor synchronizing process. Furthermore, such configurations deal with more involved post-processing computations and hence are less recommendable for dedicated hardware setups regarding autonomous vehicles. Recently developed LIDAR scanners are unaffected of such limitations and their market price has noticeably decreased within the last years. In light of these facts, LIDAR scanners are an indispensable part of autonomous cars and modern MAVs nowadays.

## 2.3 Calculating Organized Structured Point Cloud

Usual 3D LIDAR point clouds have inherent problems like high sparsity and irregularity. For our following experiments, we utilize an organized, structured grid (*OSP*) to handle these limitations. LIDAR points are typically characterized in polar coordinates as mentioned above. However, the provided dataset of Geiger et al. [10] and Behley et al. [3] provide point clouds in the 3D cartesian space exclusively. We explicitly exploit the cartesian representation to retrieve spherical coordinates when calculating the *OSP*. Hence, Equation 2.3.1 describes the relation between the cartesian representation  $\mathbf{p} = (x, y, z)^\top$  of received LIDAR and the corresponding polar coordinate  $\mathbf{p}_{pol} = (\phi, \theta, d)^\top$ :

$$\begin{aligned} x &= d \sin \phi \cos \theta, \\ y &= d \sin \phi \sin \theta, \\ z &= d \cos \phi. \end{aligned} \tag{2.3.1}$$

Besides, each point  $\mathbf{p}$  belongs to a specific label class  $l$ . This approach focuses on the primary ground truth classes *car*, *pedestrian*, *cyclist* and considers other objects as the category *don't care*. In Section 4.1 that follows, we primarily explain the individual essential characteristic of used datasets and their label distribution. Before proceeding to examine the mechanism of the *OSP*, it is necessary to define our used convention.

A LIDAR point  $\mathbf{p}_{lid} = (x, y, z, r, d, l)^\top \in \mathbb{R}^6$  summarizes all related features and corresponds to an unambiguous LIDAR point cloud  $L_t$  which denotes a set of points taken at time  $t$  from a Velodyne HDL-64E scanner.

$r$  denotes the reflectance strength of the beam and  $d$  the distance to the explored object as we already outlined in Section 2.1. The organized structured point cloud

## 2 Foundation

(*OSP*) consists of a 2D plane of height  $h \in \mathbb{N}$  and width  $w \in \mathbb{N}$ , where each pixel stores a 6D feature channel. This channel is accessible by an explicit index tuple  $(i, j) \in H \times W := \{0, \dots, h - 1\} \times \{0, \dots, w - 1\}$ . Especially, index  $i$  defines the vertical and  $j$  the horizontal position of a specific channel  $C(i, j) \in \mathbb{R}^6$  stored in the *OSP*.

The unstructured sparse point cloud is illustrated in Figure 2.2(a) before the transformation where we subsequently transform each point by

$\xi(\psi(\mathbf{p})) = (i, j)$  to an unambiguous index tuple  $(i, j)$  and accordingly storing its features in the corresponding 6D channel  $C(i, j)$ . This mapping procedure enables us to retrieve a dense structured point cloud as visualized in Figure 2.2(b) where the normalized depth information  $d$  is shown.

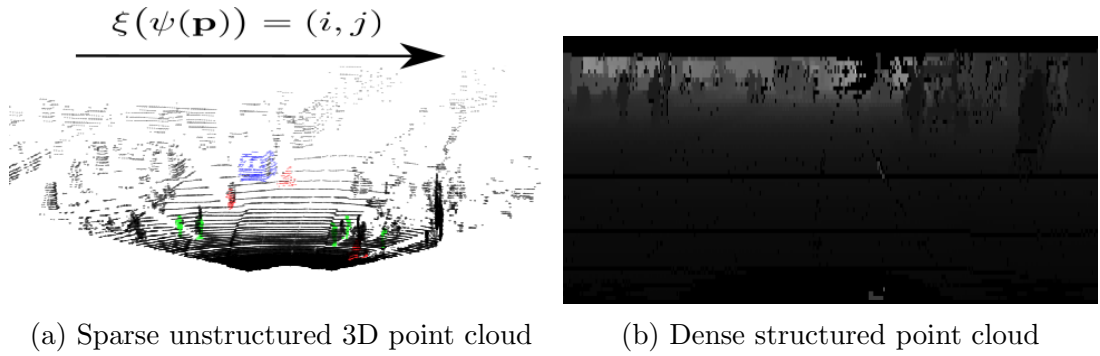


Figure 2.2: Process of point cloud transformation.

Our primary objective is to map each  $\mathbf{p}_{lid} \in L_t$  to a spherical tuple  $(\phi, \theta) \in \Phi \times \Theta \subset \mathbb{R}^2$  and afterward project such angles to an unambiguous index tuple  $(i, j)$ . As a result, we can initialize the channel  $C(i, j)$  with the features of  $\mathbf{p}_{lid} = (x, y, z, d, r, l)^\top$ . In the following, we will go into more detail regarding this projection and describe this mechanism in several steps:

First, the algorithm calculates for each  $\mathbf{p}_{lid} \in L_t$  the corresponding zenith angle  $\phi$  and azimuth angle  $\theta$  by  $\psi$  defined as:

$$\begin{aligned} \psi: \mathbb{R}^3 &\rightarrow \Phi \times \Theta \\ \psi(\mathbf{p}) &= \left( \arctan2(y, x), \arcsin\left(\frac{z}{\sqrt{x^2 + y^2 + z^2}}\right) \right). \end{aligned} \quad (2.3.2)$$

This 2D spherical projection only consults the cartesian vector  $(x, y, z)^\top \in \mathbb{R}^3$  of each  $\mathbf{p}_{lid} \in L_t$  to encode the spatial information in our *OSP*. Second, we utilize  $\xi$  defined in Equation 2.3.3 and Algorithm 1 to map the angle tuple  $(\phi, \theta)$  to an

explicit index position  $(i, j)$ :

$$\xi: \Phi \times \Theta \rightarrow H \times W. \quad (2.3.3)$$

---

**Algorithm 1:**  $\xi(\phi, \theta)$ 


---

**Data:**  $(\phi, \theta) \in \Phi \times \Theta$

**Result:**  $(i, j) \in H \times W$

**begin**

    sort  $\Phi$  in ascending order

    create a partition  $\tilde{\Phi}_{|H|} \leftarrow \bigcup_{k=1}^{|H|} \Phi_k$ , with  $|H|$  disjoint subsets  $\Phi_k \subset \Phi$

    sort  $\Theta$  in ascending order

    create a partition  $\tilde{\Theta}_{|W|} \leftarrow \bigcup_{l=1}^{|W|} \Theta_l$ , with  $|W|$  disjoint subsets  $\Theta_l \subset \Theta$

$res_i \leftarrow \emptyset$

$res_j \leftarrow \emptyset$

**for**  $\Phi_k \in \tilde{\Phi}_{|H|}$  **do**

**if**  $\phi \in \Phi_k$  **then**

$res_i = k$

**for**  $\Theta_l \in \tilde{\Theta}_{|W|}$  **do**

**if**  $\theta \in \Theta_l$  **then**

$res_j = l$

**return**  $(res_i, res_j)$

---

This allows us afterward to perform an assignment of  $\mathbf{p}_{lid}$  at the pixel location  $C(i, j)$ .  $\xi$  is a surjective function and hence comes with the complication that in some cases our *OSP* algorithm projects the small subset

$$L_{t_{ij}} := \left\{ p \in L_t \mid \xi(\psi(\mathbf{p})) = (i, j) \right\} \quad (2.3.4)$$

to one position  $(i, j)$  where  $|L_{t_{ij}}| > 1$ . However, this raises the difficulty that each pixel location  $C(i, j)$  is limited to store exactly one point  $\mathbf{p}_{ij} \in L_{t_{ij}}$  with its corresponding features. Hence, we proceed the channel assignment extremely cautious to ensure that our grid includes as many low-frequency labels as possible and prefer points closer to the sensor setup.

In the first step, our algorithm collects all affected sets  $L_{t_{ij}}$  where  $|L_{t_{ij}}| > 1$ . Next, it analyzes each occurring ground truth label  $l$  of  $\mathbf{p}_{ij} \in L_{t_{ij}}$ . Depending on  $l$ , the algorithm prioritizes to initialize the cell  $C(i, j)$  to any LIDAR point  $\mathbf{p}_{ij}$  when  $l$  is

unequal to the category *don't care*. If its the case that all points  $\mathbf{p}_{ij}$  merely relate to the *don't care* category, the algorithm assigns  $C(i, j)$  to features from a specific point  $\mathbf{p}_{\min(d)} \in L_{t_{ij}}$  which represents the lowest distance to the sensor setup. The set of index tuples containing an empty channel for a current point cloud  $L_t$  after the calculation of the *OSP* is defined as:

$$I_{t_{empty}} := \left\{ (i, j) \in H \times W \mid \forall \mathbf{p} \in L_t \xi(\psi(\mathbf{p})) \neq (i, j) \right\}. \quad (2.3.5)$$

The principle  $|I_{t_{empty}}| \geq 1$  applies for every calculated frame and hence underlines the limitations of the *OSP* representation. However, this grid yields a more compact and dense representation compared to the sparse point cloud.

## 2.4 Description of Image and LIDAR Motion

A popular method to describe dynamic changes between adjacent frames is to calculate optical flow. This area is well explored for images, and there is a large number of innovative approaches for determining optical flow [5, 8, 17]. In this section, we focus primarily on the linearised definition to describe the main aspects of this motion. One can observe that the intensity value of a specific pixel location  $(u, v)$  regarding an ordered image sequence changes over time. The corresponding intensity values at  $(u, v)$  at image  $I_{t-1}$  taken at  $t - 1$ , moved their position with the translation vector  $\mathbf{g} = (e, f) = \left( \frac{\partial u}{\partial t}, \frac{\partial v}{\partial t} \right)$  to

$$I(u, v, t - 1) = I(u + e, v + f, t - 1 + \Delta t) \quad (2.4.1)$$

at present time  $t$ . Optical flow methods try to estimate such translation vector  $\mathbf{g}$  in order to describe the overall motion from  $I_{t-1}$  and  $I_t$  by considering the time duration  $\Delta t$ . Difficulties arise, however, when calculating the non-linearised displacement field as reported by Brox et al. [5].

This specific motion is only valid under linear displacement changes and relies on the assumption of constant gray values for temporal correspondences as shown by Horn and Schunck [15]:

$$\frac{\partial I}{\partial u} e + \frac{\partial I}{\partial v} f + \frac{\partial I}{\partial t} = 0. \quad (2.4.2)$$

This brightness constancy assumption is invalid in most scenarios since the object surface brightness varies as a function of time. Based on the principle of optical flow, Scene-flow [14, 22] characterizes the translation in 3D space. Another approach to depict the 3D movement is rigid motion [13], which assumes that ad-



adjacent point clouds change uniformly. More precisely, this criterion is rooted in the premise that the relative distance, as well as the position between points, remain unaltered in contrast to Scene- or optical flow. The methodology Section 4.2 describes the mechanism to retrieve this motion from the inertial measurement unit (IMU) and GPS measurements. In our experiment, we exploit the 3D motion in the point cloud in order to obtain the 2D translation vector in our *OSP* frame. This temporal information allows us to warp previous activation maps of the CNN and *OSP* frames to the current representation.

This chapter began by describing the principle of conventional LIDAR scanner and outlined the limitations as well as the advantages. Moreover, it argued that the usage of *OSP* representations offers benefits like compactness and applicability of standard image-based CNN approaches. Finally, we took a closer look at the description of motion in images and LIDAR scans.



## 3 Related Work

The purpose of this chapter is to review the literature on recent works related to LIDAR Segmentation as well as approaches considering motion in between input frames. It begins by introducing different methodologies for preprocessing LIDAR data and segmenting such representation. We already presented various motion descriptions for image and LIDAR scan pairs in Section 2.4. Furthermore, we present existing approaches which exploit such time depending features in the similar field of LIDAR segmentation. We then proceed with a specific LIDAR Segmentation CNN called *SqueezeSeg* [30] and describe its main characteristics in detail. Building upon these results, we outline the limitations of this architecture and afterward introduce the reader to our method.

### 3.1 LIDAR Segmentation

To date, various approaches address 3D LIDAR point cloud segmentation using different strategies in data preprocessing and pipeline design. Moosmann, Pink, and Stiller [23] utilize the 3D point cloud representation in conjunction with a local convexity measure in a graph-based optimization. This method achieves good segmentation results even in non-flat areas. In contrast Hackel et al. [12] present a 3D-CNN architecture which consults five different transformed voxel resolutions for each scan, in order to output the class conditional probabilities of urban objects. The main limitation of voxel representations, however, is that LIDAR scans are highly sparse and irregularly shaped in general. Furthermore, the transformation into a cube representation leads to a high number of empty voxels which causes an excessive computational cost compared to 2D based networks. To handle such drawbacks, recent works [30, 7, 29] propose to precalculate dense grid representations which are then segmented by 2D based pipelines. The CNN *SqueezeSeg* by Wu et al. [30] utilizes additionally efficient custom architecture modules to address real-time segmentation for LIDAR point clouds. This approach becomes a central focus in the following Section 3.3.

Instead, *PointSeg* by Wang et al. [29] based their work on [30] and apply several ideas of RGB semantic segmentation methods into the CNN structure. A

slightly different, but well-researched area is LIDAR-based object detection. Its primary aim is to predict object-oriented bounding boxes in point clouds. This area has noticeably improved during the last years. Recent approaches [20, 19] demonstrate remarkable performance, proposing to exploit LIDAR scans as well as corresponding RGB images to benefit from extended input features. Simon et al. [25] outperform previous approaches in the KITTI Bird’s Eye View Benchmark [10] by utilizing LIDAR inputs solely in a fully connected network.

## 3.2 Leveraging Temporal Information

The extraction and exploitation of short-time information related to LIDAR scans is scarcely investigated at this moment. One of the few exceptions is presented by Vaquero, Sanfeliu, and Moreno-Noguer [27], which only consults range data to detect the motion vector of dynamic moving vehicles, by embedding additional information such as motion and semantic prior based on LIDAR inputs. Currently, there is no ground truth available for the optical flow equivalent on LIDAR scans. Hence, they propose a learned LIDAR-based approach called lidar flow where correspondences are obtained from optical flow via projection of point clouds. A different approach was taken by Dewan, Oliveira, and Burgard [7]. Similar to *SqueezeSeg*, they propose a mechanism for projecting each LIDAR beam into a 2D spherical plane where each channel yields three features. From this, a CNN estimates a point-wise objectness score to distinguish between movable and immovable points. A Bayes filter combines the objectness score with a dynamicity score of their previous approach [6].

Related research fields like image segmentation (IS) and visual odometry (VO) show well-suited approaches for taking advantage in extracting motion according to a chronological frame sequence, in contrast to LIDAR-based segmentation. Recent work on image segmentation by Gadde, Jampani, and Gehler [9] introduce a warping of the previous image activation with the assistance of optical flow. Moreover, they exploit this transformed representation to combine it with the current image feature activation map. Their evaluation shows a noticeable improvement by marginal runtime increase in comparison with the single frame based frame methodology. Instead, Ma et al. [21] enforce multiview consistency. They recommend warping CNN feature maps and semantic predictions of multiple views into a reference view given known image poses. Results show that exploiting multiple views outperforms single view predictions. Another work by Ummenhofer et al. [26] facilitate unconstrained pairs of images to compute depth and the corresponding camera motion.

Historically the camera ego-motion within image sequences (VO) has been tackled with hand-crafted algorithms. A currently very active research topic is learning VO. One example stems from Wang et al. [28] whom concatenate two-time correlated images and extract geometrical features through a CNN before feeding them through a recurrent neural network (RNN). In particular, they employ a long short-term memory recurrent unit to learn sequential patterns.

### 3.3 SqueezeSeg

We base our approach on the CNN called *SqueezeSeg* developed by Wu et al. [30], which takes as input a compact *OSP* representation of a 3D LIDAR point cloud as illustrated in Section 2.3. This architecture outputs a point-wise label map of common urban objects to address real-time applications such as autonomous driving. Besides, a Conditional Random Field (CRF) [31] reformulated as a recurrent unit further refines the probability output of the CNN. Moreover, this architecture yields its superior efficiency to the fact, that the implementation contains a derivation of *SqueezeNet* (CNN) presented by Iandola et al. [16]. This special CNN reaches with fifty times fewer trainable parameters the same accuracy as the widely used *AlexNet* developed by Krizhevsky, Sutskever, and Hinton [18]. In the next section, we analyze the efficient modules of *SqueezeNet* and outline the differences according to *SqueezeSeg*.

#### 3.3.1 Custom Convolutional Modules

The *FireModule* concept was introduced for the first time in *SqueezeNet*. A vital characteristic of this layer is to extract large activation maps with fewer parameters and hence smaller computational cost compared to components included in *AlexNet*. Overall, the *FireModule* is structured into 2 sub-units as illustrated in Figure 3.1. The first module called *squeeze* layer, consists of a convolutional layer which utilizes a  $1 \times 1$  filter yielding nine times fewer parameters compared to a standard  $3 \times 3$  filter.  $s1$  denotes the number of utilized  $1 \times 1$  filters of the *squeeze* layer.

Afterward, the output channels are fed into an *expand* module. This unit is structured into two convolutional layers. The first convolutional layer consists of  $e1$  times  $1 \times 1$  filters, while the latter comprises  $e3$  times  $3 \times 3$  filters. In general, the number of parameters of a convolutional layer depends on the input channels, number of filters as well as the kernel size. The authors propose to downsample included layers as late as possible to keep feature maps at high resolution to obtain highly accurate predictions.

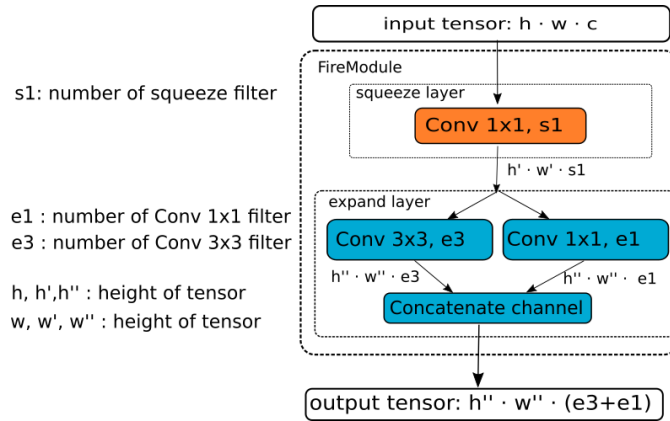


Figure 3.1: Structure of the *FireModule* from *SqueezeNet*.

Furthermore, this model allows to select the number of filters  $s1$  in the *squeeze* and of *expand* layer ( $e1 + e3$ ) empirically or trained as hyperparameters. The authors strongly recommend choosing  $s1 < (e1 + e3)$  in order to keep the number of input channels of *expand* layers low and hence reduce the overall amount of parameters by a considerable factor.

### 3.3.2 CNN Architecture

The network structure of *SqueezeSeg* is shown in Figure 3.2. The layers *Conv1a* to *Fire9* are adopted from *SqueezeNet*. At the beginning the pipeline is fed with an *OSP* of dimension  $h \cdot w \cdot c$  into the first layer *Conv1a*.

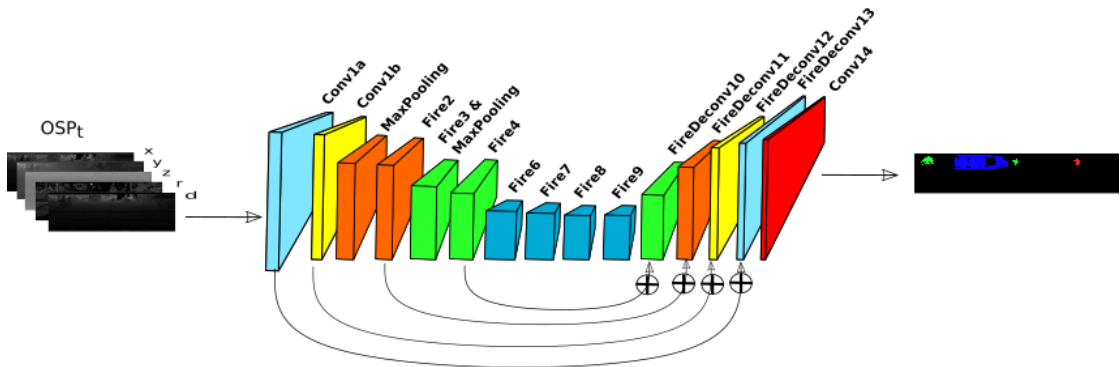


Figure 3.2: Network architecture of *SqueezeSeg*.

Afterward, several convolution and *max-pooling* operations downsample this input in order to reduce input dimension. While *SqueezeNet* downsamples in vertical and horizontal dimension, Wu et al. decided to reduce only the horizontal shape, keeping the original height in the entire segmentation process since the width of

an *OSP* is much larger than its height. In view of efficient feature map extraction, they embed eight custom *FireModules* as illustrated in Figure 3.3.

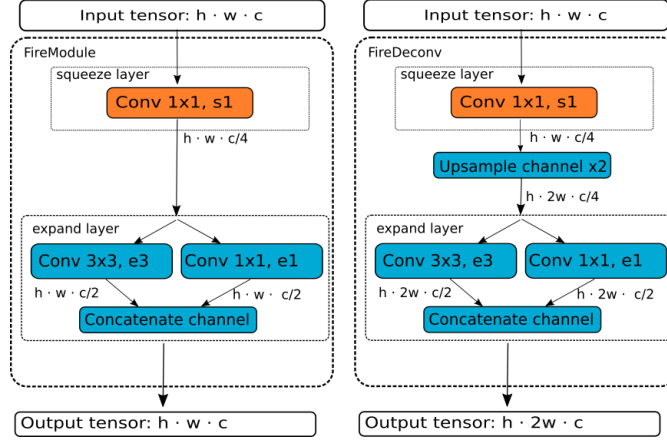


Figure 3.3: Structure of a *FireModule* (left) and a *FireDeconv* (right).

These modules take as input a tensor of size  $h \cdot w \cdot c$  and output the same dimension size. In particular, the *squeeze* layer minimizes the number of input channels by limiting the number of filters to  $s1 = \frac{c}{4}$ . Considering subsequently the *expand* layer, they decided to set the number of the  $1 \times 1$  as well as the  $3 \times 3$  convolutional filters to  $e1 = e3 = \frac{c}{2}$ . The *FireModule* concatenates the internal *expand* layer output channels which result in the original input tensor size  $h \cdot w \cdot c$ . In comparison, a conventional  $3 \times 3$  convolutional filter will require  $9c^2$  parameters combined with its computational cost of  $9hwc^2$ . The *FireModule* needs only  $\frac{3}{2}c^2$  weights and is limited to  $\frac{3}{2}hwc^2$  computations. In this case, it means that the *FireModule* uses six times fewer parameters and computations in contrast to a conventional layer yielding a  $3 \times 3$  filter. To obtain a full resolution probability map, the authors utilize a custom designed *FireDeconv* to upsample the width dimension of the activation maps by an additional factor of 2. The structure of the *FireDeconv* is similar to the *FireModule* and owns an additional deconvolutional layer between the *squeeze* and *expand* unit. In view of the output resolution of  $h \cdot 2w \cdot c$ , these layers employ  $\frac{7}{4}c^2$  parameters and need  $\frac{7}{4}hwc^2$  computations. It should be noted that a standard  $1 \times 4$  deconvolutional layer contains  $4c^2$  parameters and requires  $4hwc^2$  computational steps. Instead, the *FireDeconv* needs 2 times less parameters which is an outstanding efficiency. As the network causes spatial information loss during *max-pooling* operations, it exploits additional *skip connections* to add upsampled feature maps to lower-level feature maps of the same shape. The layer from *Conv1a* to *FireDeconv13* employ the rectified linear unit (ReLU) activation function. It outputs zero if the input is less than zero and otherwise returns the raw input. At the end, *Conv14* utilizes the *softmax* activation to output the full resolution

probability map. This class-wise label output tends to have blurry boundaries caused by downsampling operations of the *max-pooling* layer. Wu et al. decided to refine the generated probability output of *Conv14* by a CRF, reformulated as a RNN unit. This recurrent unit belongs to the class of statistical modeling methods. Applications like pattern recognition and machine learning usually utilize this kind of model. A central principle of the CRF module constitutes that it considers neighboring LIDAR features in order to assign a specific point to a probability class. It is quite likely when several neighboring points share similar features that they belong to the same class label. We refer the readers to [30, 31], for a detailed description of this module, since our primary objective in this thesis is to develop extensions related to the CNN whereas the CRF is out of the scope.

#### 3.3.3 Limitations of SqueezeSeg

There are two main reasons, why this segmentation pipeline of *SqueezeSeg* demonstrates specific limitations:

One major drawback of this approach is that the architecture uses single frame inputs to predict the class probability map of the current point cloud and does not exploit the contextual information between several adjacent scans. For instance, Geiger, Lenz, and Urtasun [11] and Behley et al. [3] provide global ego-motion estimates which allow us to retrieve the point-wise rigid motion between time correlated point clouds. As mentioned earlier, researchers [9, 21, 27, 7] already showed that leveraging semantic and motion priors in similar fields have the potential to improve results. *SqueezeSeg* is limited to a horizontal  $81^\circ$  degree FoV which is insufficient for the estimation for a potential  $360^\circ$  point cloud. Applications such as autonomous driving or MAV explorations demand efficient algorithms for segmenting the entire FoV. Thanks to recent work by Behley et al. [3], there is the opportunity to evaluate *SqueezeSeg* on complete point-wise labeled 3D LIDAR scans.

These drawbacks raise the question of how to extend *SqueezeSeg* to handle the mentioned limitations. The following chapter covers different aspects of this topic.



## 4 Overview of Methodology

The purpose of this section is to introduce the reader to our three different strategies, trying to handle the limitations of *SqueezeSeg* outlined in Section 3.3.3. Before proceeding to examine these mechanisms, it is necessary to introduce the main principals of used datasets and their linked preprocessing to enable our subsequent extensions. Furthermore, we take a close look at the two employed datasets and highlight their main characteristics as well as the label distribution.

### 4.1 Datasets

The KITTI Vision Benchmark Suite published by Geiger et al. [10] provides a vast data collection which consists of LIDAR scans, gray-scale and color images as well as the corresponding GPS\IMU sensor values. These recordings include various scenes of city traffic, residential and campus areas. However, in this thesis, we confine ourselves to the LIDAR input and corresponding GPS\IMU measurements. The LIDAR raw data collection stems from a Velodyne HDL-64E laser scanner, which explores the full horizontal FoV with 64 vertical beams. Besides, the KITTI recording setup limited the frequency of this LIDAR scanner to 10 Hz to ensure robustness for the sensor synchronization process. The LIDAR data is preprocessed and provided as a 3D point cloud with cartesian coordinates  $\mathbf{p}_{lid} = (x, y, z)^T \in \mathbb{R}^3$ . In order to retrieve the *OSP* representation, we utilize the chained function  $\xi(\psi(\mathbf{p}_{lid}))$  as described in Equation 2.3.3, to determine the corresponding position. The range value  $d$  is obtained as the euclidean distance:

$$d = \sqrt{x^2 + y^2 + z^2}. \quad (4.1.1)$$

We employ two different point cloud annotations in total which base on the KITTI measurement as highlighted above. Geiger et al. [10] provide the object oriented bounding boxes (OBB) for each point cloud to retrieve the ground truth annotation. Unfortunately, these solely depend on the section that overlaps with the front camera's FoV and hence include unnatural truncated ground truth information due to the horizontal image edges.

Theses annotations offer in our experience only for a frontal opening angle of  $81^\circ$

reliable ground truth information. Hence, we restrict the width of our *OSP* frames accordingly. The OBB may contain other points that do not belong to the object itself. Instead, a very recent work by Behley et al. [3] provides the point-wise annotations in the full FoV. This allows us to evaluate all extensions also on the full  $360^\circ$ .

Our preprocessing is nearly identical for both annotated datasets. In the next sections, we take a close look at the properties of both datasets. This section closes with the mechanism to retrieve the 6D rigid transformations.

### 4.1.1 The KITTI Vision Benchmark

#### Label Distribution

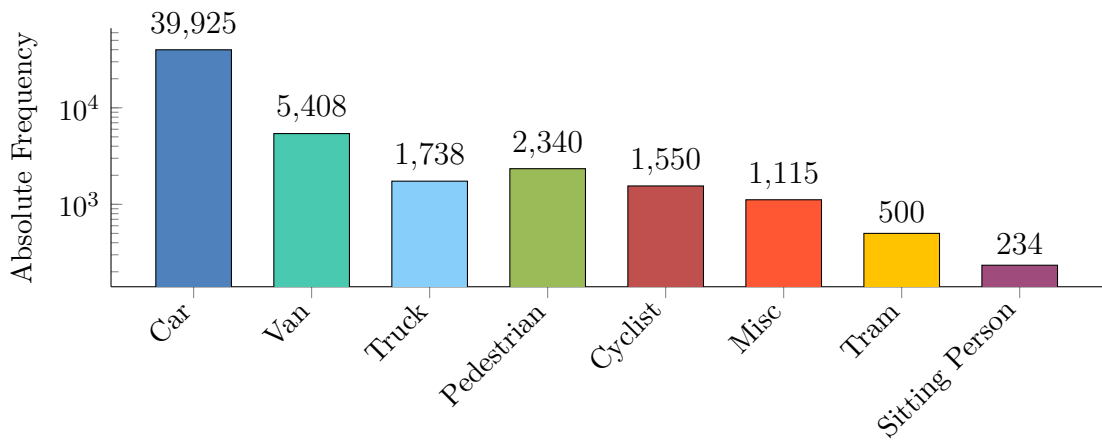


Figure 4.1: Total number of object detections.

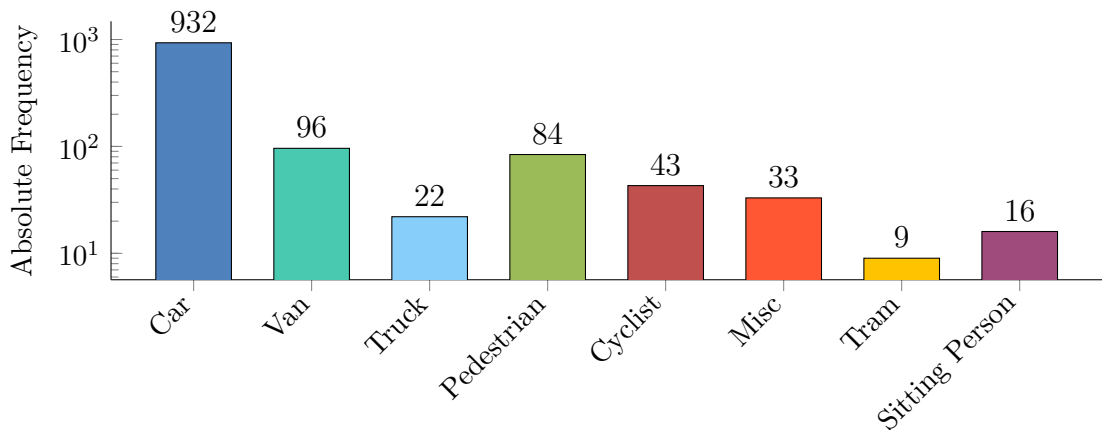


Figure 4.2: Total number of object instances.

The label collection of the KITTI Vision Benchmark consists of ordinary urban

objects like *pedestrian*, *cyclist*, *sitting person*, *car*, *van*, *truck*, *tram*, or *misc*. We only consider scene sequences where the corresponding OBB is available. Each object belongs to an unambiguous 3D oriented bounding box which is unfortunately limited to  $81^\circ$  FoV. However, this dataset does not cover every object in the point cloud, especially in cases when such classes are in the background. Consequently, there are some points interpreted as the *don't care* category according to our *OSP*, even though they are part of a specific object. Furthermore, the number of objects per class is unevenly distributed as visualized in Figure 4.1. Obviously, each object may occur multiple times in subsequent scans. Figure 4.2 shows the number of unique instances per class and demonstrates a similar unequal distribution. In general, the amount of reflected points on *cars* or *vans* is much higher compared to thin objects like *cyclist* or *pedestrian*. In the scope of this thesis, we focus on the primary ground truth classes *car*, *pedestrian*, and *cyclist* and decided to include *trucks* and *vans* to the label *car*. Besides the fact that we distinguish between *cyclist* and *pedestrian*, it is less reasonable to denote *sitting persons* to one of the three-aspected labels either. The classes of *tram*, *misc* and *sitting person* also demonstrate the lowest frequency of occurrence and rarely share geometrical features as well as behavior patterns in comparison to our primary classes. Subsequently, we assign all points which do not belong to any of the ground truth labels as *don't care*.

### Determination of Ground Truth Utilizing Oriented Bounding Boxes

The determination of the point-wise ground truth takes two intermediate steps, since the OBBs are given in the camera frame.

First, we apply  $T_{cam}^{lid} \in \mathbb{R}^{4 \times 4}$  to every point  $\mathbf{p}_{lid}$  of a specific point cloud  $L_t$  taken at time  $t$ . This step allows us to move  $\mathbf{p}_{lid}$  from LIDAR coordinates into the camera coordinate system. The matrix  $T_{cam}^{lid}$  is also provided by Geiger et al. [10] and the operation results in a new point cloud:

$$L_{t_{cam}} := \left\{ T_{cam}^{lid} \mathbf{p}_{lid} \mid \mathbf{p}_{lid} \in L_t \text{ and } T_{cam}^{lid} \in \mathbb{R}^{4 \times 4} \right\}. \quad (4.1.2)$$

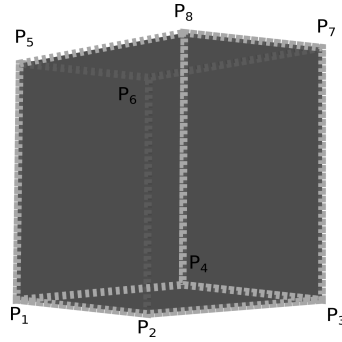
The corners of an OBB are denoted by  $\mathbf{p}_1$  to  $\mathbf{p}_8 \in \mathbb{R}^3$  as illustrated in Figure 4.3(a). The three essential directions  $\mathbf{a}$ ,  $\mathbf{b}$  and  $\mathbf{c} \in \mathbb{R}^3$  are pairwise perpendicular edges of the rectangular box and we determine them with Equation 4.1.3:

$$\begin{aligned}
 \mathbf{a} &= \mathbf{p}_1 - \mathbf{p}_2, \\
 \mathbf{b} &= \mathbf{p}_1 - \mathbf{p}_4, \\
 \mathbf{c} &= \mathbf{p}_1 - \mathbf{p}_5.
 \end{aligned}
 \tag{4.1.3}$$

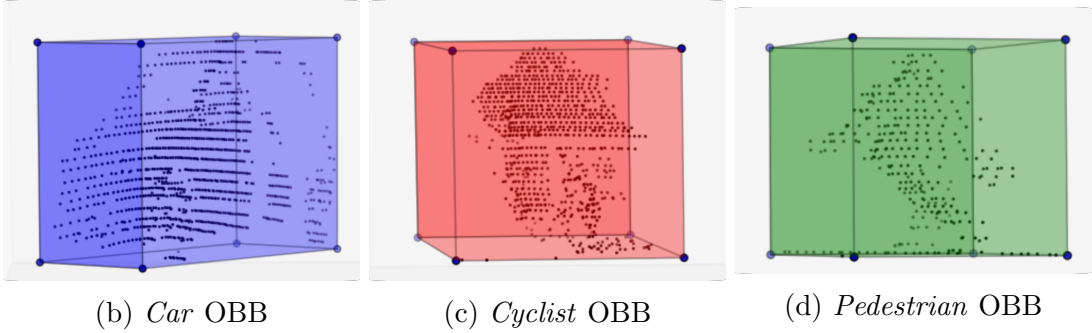
The transformed LIDAR point  $\mathbf{p}_{cam} \in L_{t_{cam}}$  lies within the OBB if and only if, each scalar product between the point  $p_{cam}$  and the corresponding direction fulfills following criterion:

$$\begin{aligned}
 &(\langle \mathbf{p}_1, \mathbf{a} \rangle < \langle \mathbf{p}_{cam}, \mathbf{a} \rangle < \langle \mathbf{p}_2, \mathbf{a} \rangle) \wedge \\
 &(\langle \mathbf{p}_1, \mathbf{b} \rangle < \langle \mathbf{p}_{cam}, \mathbf{b} \rangle < \langle \mathbf{p}_4, \mathbf{b} \rangle) \wedge \\
 &(\langle \mathbf{p}_1, \mathbf{c} \rangle < \langle \mathbf{p}_{cam}, \mathbf{c} \rangle < \langle \mathbf{p}_5, \mathbf{c} \rangle).
 \end{aligned}
 \tag{4.1.4}$$

As a result, we can retrieve the point-wise label information for each point as depicted in Figure 4.3.



(a) Representation of a standard oriented bounding box.



(b) *Car* OBB

(c) *Cyclist* OBB

(d) *Pedestrian* OBB

Figure 4.3: Ground truth assignment by oriented bounding boxes.

The major drawback of obtaining ground truth information by utilizing the OBB constitutes that it covers also some points which do not actually belong to the real object. For instance, the OBBs in Figure 4.3(c) and 4.3(d) include the ground

plane which does not represent a real part of a *pedestrian* or *cyclist* either. This characteristic might distort main features of these objects and could lead to a situation where *SqueezeSeg* might learn misinterpreted patterns. In the following, we present another dataset which closes this gap.

#### 4.1.2 A Dataset for Semantic Segmentation of Point Cloud Sequences

The recent work from Behley et al. [3] provides accurate point-wise annotations covering 22 different classes for the KITTI VO [11]. They developed an efficient labeling tool for annotation of point clouds based on simultaneous localization and mapping (SLAM) poses. Moreover, their labeling procedure demonstrates more precise annotations and considers even complex structures of various objects. This approach does not possess the problems of the OBB method outlined above.

Category	Class	<i>Car</i>	<i>Pedestrian</i>	<i>Cyclist</i>	<i>Don't care</i>
ground	road				X
	sidewalk				X
	parking				X
	other ground				X
structure	building				X
	other structure				X
vehicle	car	X			
	truck	X			
	bicycle			X	
	motorcycle			X	
	other vehicle				X
nature	vegetation				X
	trunk				X
	terrain				X
human	person		X		
	motorcyclist			X	
	bicyclist			X	
object	fence				X
	pole				X
	traffic sign				X
	other objects				X
outlier					X

Table 4.1: **Label reassignment:** Raw label of the 360° FoV dataset is assigned to our primary classes.

However, we proceed some class reassignment operations on the 360° FoV dataset, since this thesis restricts to the primary classes *car*, *cyclist* and *pedestrian* as illustrated in Table 4.1. We decided to map the label *truck* to the label *car*. Furthermore, we do not distinguish between moving and stationary objects. The category *motorcyclist* denotes a person riding the vehicle or standing nearby the vehicle and share nearly identical feature patterns as a *bicyclist*. The label remapping procedure assigns *bicycle*, *bicyclist*, *motorcycle* and *motorcyclist* to our main class *cyclist*, since we do not differentiate between a *bicyclist* and a *bicyclist* riding the bike.

### 4.1.3 Dataset Comparison

The previous chapters already outline the main characteristics of our utilized dataset. Table 4.2 highlights our selection of the training, validation and testing set for each utilized dataset. For the following extensions, *SqueezeSeg* optimize its parameter on the training set and simultaneously evaluate on a small validation set the intermediate metric results. Section 5 presents the segmentation results for all implemented extensions which base on the testing set. This chapter also illustrates the utilized metrics and computational differences between both datasets.

Dataset	OSP Resolution	Number of Scans			
		Training Set	Validation Set	Testing Set	Total Set
81° FoV	64 · 384	9732	315	2791	12838
360° FoV	64 · 1920	19130	4071	20351	43552

Table 4.2: LIDAR datasets split in training, validation and test set.

In total, the 360° FoV dataset yields roughly 16 times more annotation information compared to the limited FoV KITTI dataset because of resolution and frame quantity. Table 4.3 shows how likely it is that an arbitrary pixel belongs to one of the following categories.

Dataset	<i>Don't care</i>	<i>Empty</i>	<i>Car</i>	<i>Pedestrian</i>	<i>Cyclist</i>
81° FoV	77.6%	20.2%	3.0%	0.0045%	0.0048%
360° FoV	55.6%	41.5%	2.8%	0.002662%	0.0066%

Table 4.3: Significant dataset differences at the pixel label distribution.

One remarkable difference is that the *OSP* frames in the limited FoV consist of more pixels which belong to one of our primary classes. Notably, the category *empty* represents the probability that a randomly chosen frame index holds an

empty entry. The generated *OSP* of the complete FoV dataset yields in average the lowest valuable pixel density due to its high empty channel proportion.

## 4.2 Ground Truth Pose Retrieval from GPS and IMU Measurements

In this section, we examine the rigid motion between two LIDAR scans  $L_{t-1}$  and  $L_t$  taken at successive times  $t - 1$  as well as  $t$ , respectively. The proper rigid transformation describes the orientation and translation movement from a specific origin. Our primary objective is to retrieve such transformation to determine the LIDAR's ego-motion from  $L_{t-1}$  to  $L_t$  during the time differences  $\Delta t$ . In our case  $R_t^{t-1} \in \mathbb{R}^{3 \times 3}$  specifies the change of orientation, whereas  $\mathbf{l} = (u, v, w)^\top \in \mathbb{R}^3$  denotes the translation from origin  $L_{t-1}$  to  $L_t$ . First, we focus on the orientation  $R_t^{t-1}$ . This sensor setup provides at a given time  $t$  the orientation through the roll  $\gamma$ , pitch  $\delta$  and yaw angle  $\epsilon$ . The value of  $\gamma$  describes orientation around the  $x$  axis whereas  $\delta$  relates to  $y$  axis and  $\epsilon$  to the  $z$  axis. Let us assume that we want to transform the global pose from time  $t = 0$  to the current pose at time  $t$ . In order to accomplish that, we create in the beginning the three rotation matrices for each corresponding axis as follows:

$$R_{x_t} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{pmatrix}, \quad (4.2.1)$$

$$R_{y_t} = \begin{pmatrix} \cos(\delta) & 0 & \sin(\delta) \\ 0 & 1 & 0 \\ -\sin(\delta) & 0 & \cos(\delta) \end{pmatrix}, \quad (4.2.2)$$

$$R_{z_t} = \begin{pmatrix} \cos(\epsilon) & -\sin(\epsilon) & 0 \\ \sin(\epsilon) & \cos(\epsilon) & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (4.2.3)$$

In the next step, we want to combine above rotation matrices into one compact rotation matrix. Since matrix multiplication is not commutative, we follow the *ZXY* convention:

$$R_t^0 = R_{x_t} \cdot R_{y_t} \cdot R_{z_t}. \quad (4.2.4)$$

Subsequently, we can also analogously determine the rotation matrix  $R_{t-1}^0$  of time  $t - 1$ .

In order to obtain the translation  $\mathbf{l}$  from the origin at time  $t$ , the latitude  $\alpha$  and

## 4 Overview of Methodology

longitude  $\beta$  are provided by the GPS\IMU system. These measurements describe together the absolute position on earth. In particular, both values are represented as angles and describe coordinates in a spherical space. Let  $\mathbf{p}_{earth} = (\alpha, \beta)$  denote an arbitrary point on Earth's surface. In this case  $\alpha \in [-90^\circ, 90^\circ]$  defines the angle between equatorial plane and the normal to the surface at  $\mathbf{p}_{earth}$ . The equator ( $\alpha = 0^\circ$ ) constitutes the center, which separates South Pole ( $-90^\circ$ ) as well as North Pole ( $\alpha = 90^\circ$ ). A meridian leads from South Pole and North Pole and represent the half of an imaginary great circle on the Earth's surface. The longitude  $\beta \in [-180^\circ, 180^\circ]$  specifies the angle between prime meridian ( $\beta = 0$ ) and the meridian crossing the point  $\mathbf{p}_{earth}$ . To obtain the euclidean representation  $(u, v)$  of  $\alpha$  and  $\beta$ , we apply a cylindrical projection, called Mercator Projection on these coordinates:

$$u = s \cdot r \cdot \left( \frac{\pi \cdot \beta}{180} \right), \quad (4.2.5)$$

$$v = s \cdot r \cdot \log \left( \tan \left( \frac{\pi(90 \cdot \alpha)}{360} \right) \right). \quad (4.2.6)$$

The variable  $r \approx 6.378.137$  m is an approximation of the earth radius and  $s = \cos \left( \frac{\alpha \cdot \pi}{180} \right)$  denotes the Mercator scale as illustrated in [10].  $w$  describes the altitude measure which is independent of this projection and we apply it directly to  $\mathbf{l} = (u, v, w)^\top$ . Furthermore, the resulting pose with orientation  $R_t^0$  and origin  $\mathbf{l}$  is obtained as:

$$D_t^0 = \begin{pmatrix} R_t^0 & \mathbf{l} \\ 0 & 1 \end{pmatrix} \in \mathbb{R}^{4 \times 4}. \quad (4.2.7)$$

We finally acquire the transformation from  $t - 1$  to  $t$ :

$$D_t^{t-1} = D_t^0 \cdot (D_{t-1}^0)^{-1} = D_t^0 \cdot D_0^{t-1}. \quad (4.2.8)$$

The group of rigid transformations also includes reflections which would transform an object from the left side into the right side. In our case  $D_t^{t-1}$  is called a proper rigid transformation since  $\det(R) = 1$  and hence it does not perform any reflection transformation. We can now apply every rigid transformation  $D_t^{t-1}$  to every point in set  $L_{t-1}$  to acquire at time step  $t$  the transformed point set

$$L_t^{t-1} := \left\{ D_t^{t-1} \mathbf{p} \mid \mathbf{p} \in L_{t-1} \right\}. \quad (4.2.9)$$

The difference between  $L_{t-1}$  and  $L_t$  is apparently visible in Figure 4.4(a). In



contrast, the transformed representation  $L_t^{t-1}$  matches well with  $L_t$  as illustrated in Figure 4.4(b). The background is filtered for improved visibility.

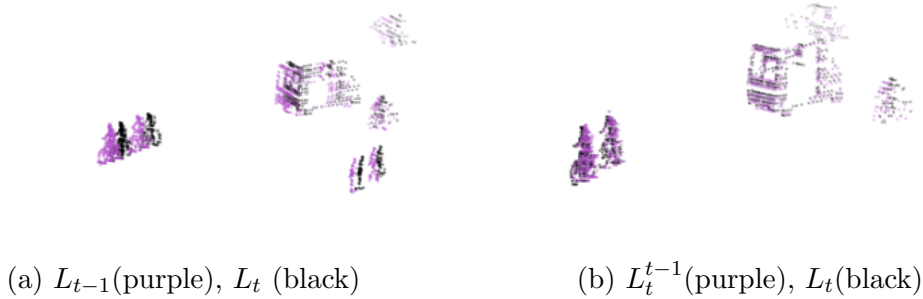


Figure 4.4: Consecutive point clouds do not match well (left). The correct transformation reduces the discrepancy between the objects (right).

### Ground Truth Pose Retrieval from SLAM

The main principle of SLAM is to build maps of the surrounding area and to determine the location of the sensor setup simultaneously. The dataset from Behley et al. [3] also provide the ground truth pose  $\tilde{D}_t^0$  which base on a SLAM procedure. The motion is obtained in the camera frame using a Surfel-based Mapping (SuMa) approach by Behley and Stachniss [4]. Hence, we apply the transformation  $T_{lid}^{cam} \in \mathbb{R}^{4 \times 4}$  from [11] to the pose  $\tilde{D}_t^0$ . This step allows us to retrieve the corresponding pose for the LIDAR point clouds:

$$D_t^0 = T_{lid}^{cam} \tilde{D}_t^0. \quad (4.2.10)$$

We apply analogously the Equation 4.2.8 to determine the ground truth rigid transformation  $D_t^{t-1}$ .

## 4.3 Experimental Approach

Recorded LIDAR data yields an unpredictable amount of reflected points since some surface structures in the surrounding area deflect reflections of transmitted beams as we illustrated in Section 2.2. As a result, most point clouds are incomplete and hence the corresponding *OSP* representations contain frequently empty feature channels. It is most likely that *SqueezeSeg* performs on partial point clouds not as good as on input representation which includes more information about the

environment. Hence, this thesis aims to evaluate different methods which utilize prior information to compensate incomplete single frames.

In total, we implemented three groups of strategies where each strategy employs a different range of prior knowledge as well as an unique methodology to embed them additionally in the network pipeline of *SqueezeSeg*. In our first extension, we extend the input channels by concatenating the feature of the previous *OSP* frame additionally. The second extension aims to transform prior point clouds into a reference view given the sensors ego-motion, and subsequently to complete empty cells of the current input frame. Moreover, the third method combines internal activation representations of the previous and the current segmentation pass. In the next section, we primarily focus on the motivation and implementation of each strategy. The following Section 5 presents the evaluation results and analyzes the impact on segmentation performance.

### 4.3.1 Extension 1: Input Robustness through Feature Concatenation

In general, one can observe that the spatial environment may not change dramatically much between adjacent LIDAR frames. It depends mainly on the recording frequency, the velocity of the sensor setup as well as on the dynamically moving objects in the environment. In our case, the LIDAR dataset was recorded with a low frame rate of 10 Hz on a moving vehicle. Even though, the environment does not exhibit significant changes in most situations. The concatenation of two adjacent frame features taken at successive times  $t - 1$  and  $t$ , is a widely used method to extract geometric features and learning sequential patterns with the assistance of a RCNN unit [28]. A prior  $OSP_{t-1}$  frame might include valuable features at a specific index location  $(i, j)$  which are missing in the current frame. Hence, we believe that a simple concatenation operation of adjacent frames might close this gap and produce a more robust input representation in contrast to the single frame approach. The following subsections introduce two different variations of this strategy.

#### Extension 1.1: Naïve Concatenation

Instead of passing single LIDAR inputs through *SqueezeSeg*, we double the number of input channels for the current input by concatenating the previous and current LIDAR scan. This operation results in an input with a size of  $h \cdot w \cdot 10$  as illustrated in Figure 4.5. We call this approach the *naïve concatenation* because it ignores the motion included 2D displacement between the two *OSPs*.

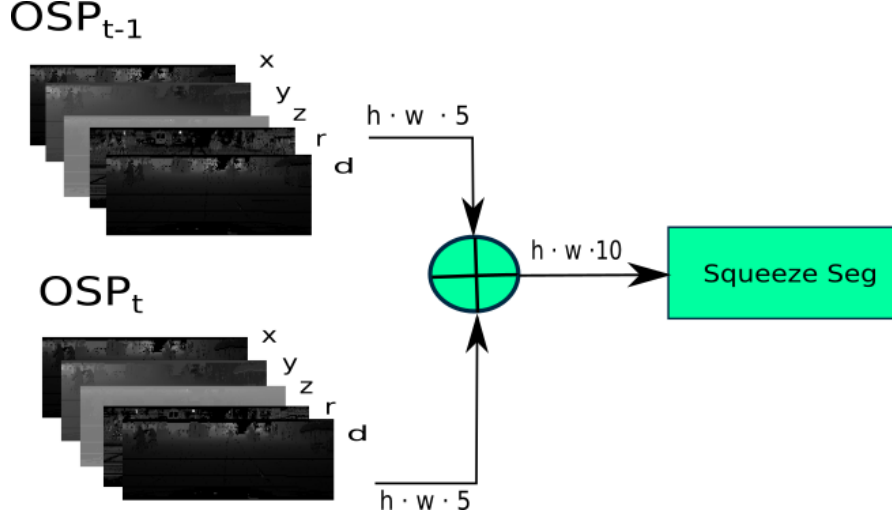


Figure 4.5: Feature channel concatenation of prior  $OSP_{t-1}$  and current  $OSP_t$ .

More precisely, the previous feature channels  $C_{t-1}(i, j)$  change the index location to  $C_t(i', j) = C_{t-1}(i + e, j + f)$  through the 2D displacement vector  $(e, f)$ . Hence, stacking the input channels assumes a direct relationship between the overlapping  $OSP$  cells. We resolve this limitation in the next section.

### Extension 1.2: Warping Concatenation

The second extension aims to establish a closer relationship between both concatenated inputs by exploiting the rigid transformation. In particular, this implementation maps each point  $\mathbf{p}$  from the previous LIDAR point cloud  $L_{t-1}$  to the current 3D LIDAR coordinate system. This operation results in  $L_t^{t-1}$  as shown in Equation 4.2.9.

In the following step, we consider the changed feature channel arrangement by warping  $OSP_{t-1}$  through recalculation of the  $OSP$  from the transformed point cloud  $L_{t-1}^t$  as described in Section 2.3. The resulting  $OSP_{t-1}^{t-1}$  also includes all 2D displacements simultaneously, and removes the need to determine the exact displacement  $(e, f)$ . Subsequently, the pipeline concatenates the feature channels of the current  $OSP_t$  together with the  $OSP_{t-1}^{t-1}$  of the corresponding point cloud  $L_t^{t-1}$ . This extension results in a small increase in computational cost in comparison to our first extension 1.1. In our opinion, it offers more valuable input information since the transformed  $OSP_{t-1}^{t-1}$  and  $OSP_t$  represents a stronger pairwise feature correlation. Figure 4.6(a) and Figure 4.6(b) illustrate the unwrapped distance channels of the first extension. One can realize by this illustration that the parking cars in the present frame  $OSP_t$  seem closer to the sensor setup com-

pared to the prior  $OSP_{t-1}$ .

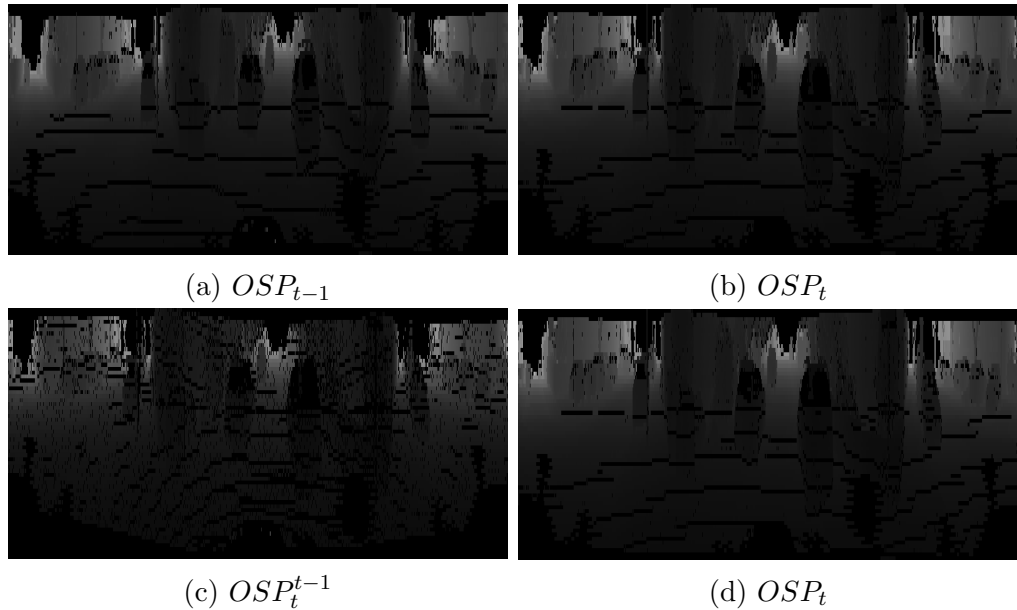


Figure 4.6: **Image warping:**  $OSP_t^{t-1}$  includes the 2D displacement between time step  $t$  and  $t - 1$  in contrast to  $OSP_{t-1}$ .

The first implementation ignores this movement during the time duration of 100 ms. Instead, the extension 1.2 considers the 2D displacement and transforms the prior frame into a new representation as illustrated in Figure 4.6(c), where each distance channel represents a stronger pairwise feature correlation regarding present frame Figure 4.6(d). Now, one can observe that the parking cars in Figure 4.6(c) and (d) are equally far away from the sensor. In particular, there are no significant 2D displacements visible.

### 4.3.2 Extension 2: Augmenting Present LIDAR Input Using Ego-motion Estimates

The strategies of extension 1 consider previous inputs by doubling the number of feature channels. As a result, the channel input complexity and the associated computational effort increases in comparison to a single frame mechanism. In the following evaluation Section 5, we analyze the parameter as well as the runtime differences more precisely. The concatenated feature maps of previous extensions exhibit at several indices positions nearly identical input. In this section, we present a data preprocessing module called Augmented Reference View (ARV) and preconnected this unit to *SqueezeSeg*. The basic principle of this strategy is to complete the empty channels of the current scan  $OSP_t$  and avoid increased

complexity of the input. First, this module transforms a specific number  $n$  of prior point clouds with the assistance of ego-motion estimates in a reference view set

$$V_t^n := \bigcup_{i=1}^n L_t^{t-i}. \quad (4.3.1)$$

The reference view set  $V_t^n$  contains all  $n$  previous point clouds transformed to the current pose of  $L_t$ . In the following, we aim to determine the corresponding index positions of the whole reference view set. Hence, we reapply the spherical projection  $\psi$  and index mapping  $\xi$  to each point  $\mathbf{p}$  of the reference view  $V_t^n$ , in order to retrieve the corresponding pixel location set

$$I_t^n := \left\{ \xi(\psi(\mathbf{p})) \mid \mathbf{p} \in V_t^n \right\}. \quad (4.3.2)$$

This index set represents all feasible candidates to complete the present  $OSP_t$ . Furthermore, we determine all empty pixel locations  $I_{t_{empty}}$  of our current  $OSP_t$  frame as defined in Equation 2.3.5. Next, the cut set  $I_{cut} = I_{t_{empty}} \cap I_t^n$  filters all final index tuple candidates which can be later utilized for completing empty entries of the  $OSP_t$ . Finally, the algorithm assigns each channel  $C_t(i, j) = \mathbf{p}$  to the corresponding point where  $\mathbf{p} \in V_t^n$  and  $\xi(\psi(\mathbf{p})) = (i, j) \in I_{cut}$ . As it is the case with our  $OSP$  projection in Section 2.3, the ARV module always prefers the nearest point from the sensor setup to augment a specific channel. Empty cells represent a distance  $d = \infty$  for the channel assignment.

In summary, the ARV module is a data preprocessing module which completes empty entries of the present  $OSP_t$ . The augmented  $OSP_t$  yields more input information in contrast to one standard frame. As a result, *SqueezeSeg* receives a more robust input representation, and hence we expect a better segmentation performance in comparison to the standard frame input. The images on the left side in Table 4.4 demonstrate the standard input before the ARV operation. Subsequently, this unit transforms points of prior point clouds into the current frame as highlighted on the right side in Table 4.4. The augmented entries in the ground truth image are highlighted in purple for better visibility. The ARV operation allows bridging gaps as shown in the corresponding point cloud in Figure 4.7.



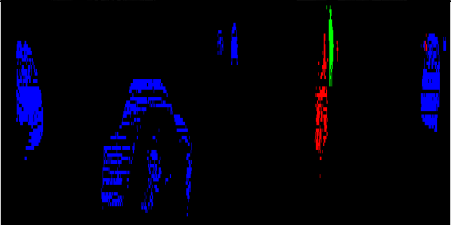
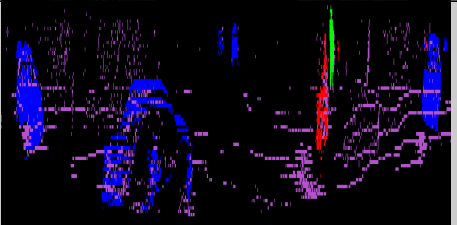
Image	Original	After ARV
range channel		
ground truth		

Table 4.4: The ARV module exploits prior point clouds to complete empty entries.

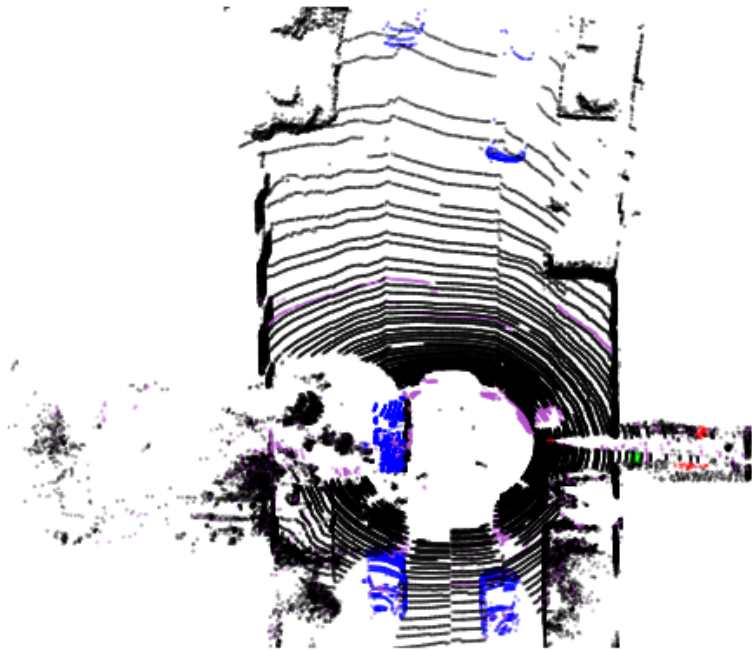


Figure 4.7: Bird's eye view after ARV with augmented points (purple).

### 4.3.3 Extension 3: Combining Successive Activations

As illustrated before, *SqueezeSeg* utilizes an additional CRF unit to refine the probability output of the CNN. However, this CRF unit has the limitation that the internal activation representations of *SqueezeSeg* are inaccessible for further refinements. For instance, the first downsampling layer extracts small activation maps and might include essential features. The approach of Gadde, Jampani, and Gehler [9] mentioned earlier proposed a method for warping previous image activations with the assistance of transformed optical flow and combine it with the current activation map to increase IS results. Inspired by this approach, we believe that consulting the current and previous activation maps of several specific layers might provide further improvements to *SqueezeSeg*. According to previous strategies, the present input representation shows several empty entries. All resulting activation feature maps depend on this input. The combination of two successive activations offers the benefit that the architecture considers different feature maps for the segmentation of the current *OSP*. We hypothesize that this additional information will have a positive impact on the perception result. As an additional technical clarification, we denote an activation map  $a_t^k$  as the output of the  $k$ -th layer from *SqueezeSeg* which takes an *OSP* $_t$  at time  $t$ . Our primary objective in this extension is to extend *SqueezeSeg* in such a way that it is capable to fuse the prior activation  $a_{t-1}^k$  with the current  $a_t^k$  activation. In particular, we aim to apply this operation to three different  $k$ -th layer as illustrated in Figure 4.8. We define in the next subsection a baseline which includes the fundamental fuse operation and subsequently show a method which warps successive activation maps.

#### Extension 3.1: Linear Activation Combination

The network architecture of *SqueezeSeg* consists of several individual modules which output activation maps on different scales. Especially, the outputs of the *max-pooling* layers represent more high-level semantics in comparison to the *Fire-Modules* *Fire5*,  $\dots$ , *Fire8*. The *skip connections* combine *DeconvModules* low-level output with more high-level features by adding both maps together. Inspired by this network structure, we want to incorporate prior and present feature maps of three specific layers, where each module yields a different level of semantic information. In order to accomplish that, we choose for our baseline the activation maps of the *Fire3*  $a_t^3$ , *Fire8*  $a_t^8$  and *FireDeconv11*  $a_t^{11}$  modules. Our developed *FuseModule*, illustrated in Figure 4.9, fuses accordingly the prior activation maps  $a_{t-1}^k$  and  $a_t^k$  by performing a weighted addition:

$$f(a_{t-1}^k, a_t^k) = w_1^k a_{t-1}^k + w_2^k a_t^k. \quad (4.3.3)$$

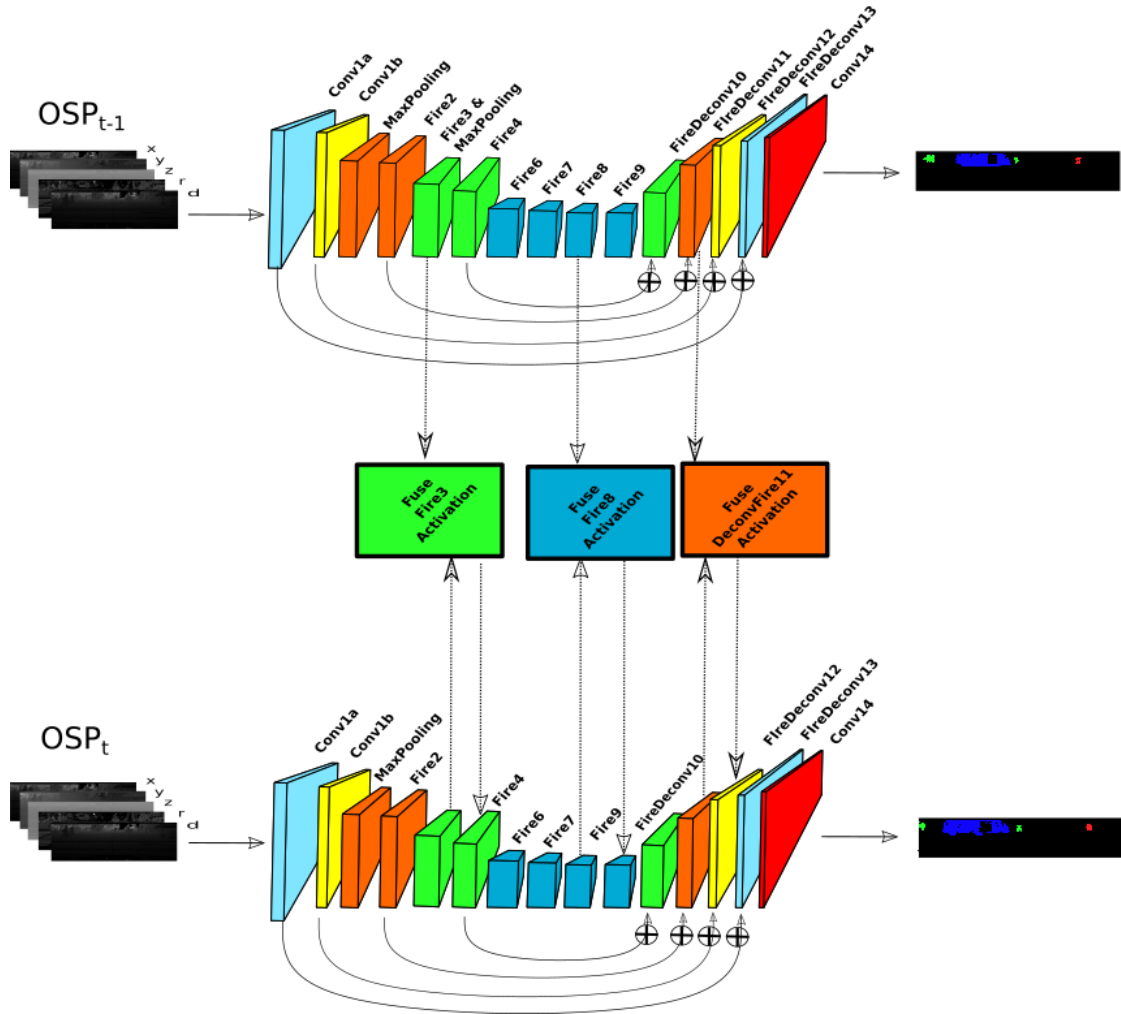


Figure 4.8: **Extension 3:** Combining successive activations through *Fuse Activation modules*.

The weight vectors  $w_1^k$  and  $w_2^k$  are of equal length as the corresponding feature channel size of the activation map. This method should ensure that *SqueezeSeg* combines and learns only essential features. Furthermore, we update the weights in every training iteration through backpropagation, since this implementation is fully differentiable.

### Extension 3.2: Activation Warping Using Ego-Motion Estimates

The primary purpose of this section is to introduce a method to extract 2D displacements in our *OSP* by transformation given an ego-motion estimate. Our following extension utilizes this information for warping activation sequences.



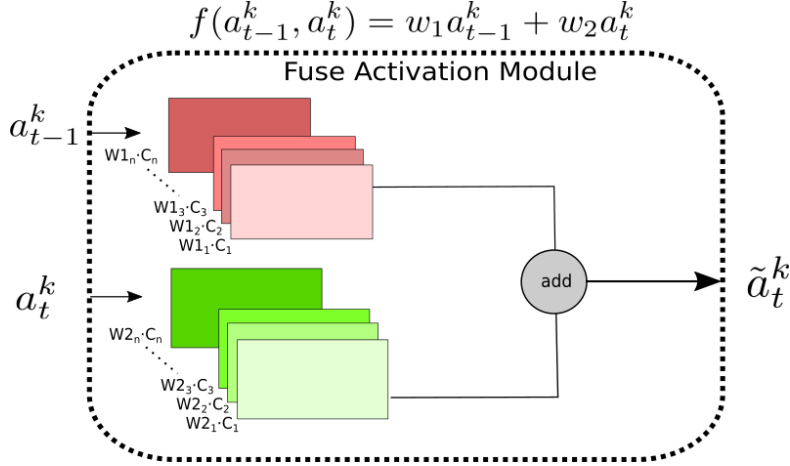


Figure 4.9: **Fuse Activation Module:** Fuses prior and present activations.

### Extraction of 2D Displacement Maps

One arguable weakness of optical flow for this specific application is that this brightness constancy assumption is inapplicable for LIDAR data since the reflectance measurement  $r$  demonstrates an unreliable feature as mentioned in Section 2.2. Besides, the authors of [5] also report that the linear optical flow assumption is not valid in most cases and especially not for large displacements. With this in mind, this linear criterion is not fulfilled by LIDAR scans obtained with a frequency of 10 Hz onboard a moving vehicle. As a result, we utilize the 6D rigid transformation for extraction of displacement indices between an  $OSP_{t-1}$  taken at time  $t - 1$  and the following  $OSP_t$  recorded at moment  $t$ , with two intermediate steps:

First, we map each point  $\mathbf{p} \in L_{t-1}$  by  $D_t^{t-1}$  to obtain the point cloud representation  $L_t^{t-1}$  at the current time step using Equation 4.2.9. This rigid transformation includes the translation as well as orientation change in 3D space between  $L_t$  and  $L_{t-1}$ . Afterward, we consider additionally the changed index arrangement between  $OSP_t$  and  $OSP_t^{t-1}$  by projecting point cloud  $L_t^{t-1}$  to the  $OSP_t^{t-1}$  representation. Next, we compare the previous point index location  $(i, j) = \xi(\psi(\mathbf{p}))$ ,  $\mathbf{p} \in L_{t-1}$  with the index location  $(i', j') = \xi(\psi(D_t^{t-1}\mathbf{p}))$  of corresponding point  $D_t^{t-1}\mathbf{p} \in L_t^{t-1}$  in order to retrieve the index displacement  $(\tilde{i}, \tilde{j}) = (i, j) - (i', j')$  between our  $OSP_{t-1}$  and  $OSP_t$ . Next, we aggregate the index displacements between both frames through a displacement field  $S_{t-1}$  where each cell  $s_{ij}$  stores the displacement  $(\tilde{i}, \tilde{j})$ . In contrast to optical flow, our displacement field  $S_{t-1}$  only considers the spatial change of the moving vehicle and thus does not contain the individual movement of dynamic objects in the surrounding area. Upon closer observation of the KITTI dataset, we experienced that in average the amount of dynamic moving objects

is considerably low compared to the number of static objects like parking cars or standing pedestrians. The following section describes explicitly the mechanism to exploit this 2D motion according to prior  $OSP_{t-1}$  to warp the corresponding activation map in to the present representation.

### Activation Warping Using 2D Displacement Maps

In our point of view, the previous extension 3.1 does not perform a fully appropriate feature mapping operation, since it ignores the changed feature-pixel arrangement caused by the time duration between  $t - 1$  to time  $t$ . In this section, we close this gap by warping the prior activation into the current representation. Gadde, Jampani, and Gehler [9] proposed to map the pixel location  $(x, y)$  of a prior image activation to the corresponding index  $(x', y')$  at the present time by considering the corresponding optical flow  $(\tilde{e}, \tilde{f})$ . This motion form is unsuitable for the  $OSP$  frames as we outlined in the section before. As a result, we calculate for prior  $OSP_{t-1}$  a 2D displacement field  $S_{t-1}$  with the assistance of an ego-motion estimate as described in Section 4.3.3.  $(\tilde{i}, \tilde{j})$  stored in channel  $s_{ij}$  at  $S_{t-1}$  denotes the displacement difference between the channel  $C(i, j)$  of prior  $OSP_{t-1}$  at position  $(i, j)$  to its corresponding location  $(i', j') = (i + \tilde{i}, j + \tilde{j})$  at current time step  $t$ . We then aim to warp a specific input channel  $C(i, j)$  of prior input  $OSP_{t-1}$  to the corresponding position at the current time step  $t$ . The warp mechanism allows us to use the displacement vector  $s_{ij} = (\tilde{i}, \tilde{j})$  to map the feature channels  $C(i, j)$  to  $C(i + \tilde{i}, j + \tilde{j})$ . Furthermore, we exploit this logic to warp  $a_{t-1}^k$  to the next representation  $\hat{a}_t^k$ . However, each activation map consists of different resolution sizes compared to our displacement map  $S_t$  caused by the downsampling operation of the *max-pooling* layer. For this reason, we have to squeeze individually  $S_t$  to the same width dimension for each activation map. Let  $w_{a^k}$  denote the time independent width of the activation map  $a^k$  and the width of  $S_{t-1}$  as  $w_s$ . In this extension, we decided to choose every  $\Delta w = \frac{w_s}{w_{a^k}}$  column of  $S_{t-1}$  and collected them in a smaller displacement map  $S_{t-1}^k$  with a horizontal size of  $w_{a^k}$ . Besides, the warping algorithm normalizes the horizontal displacement of  $S^k$ . Let  $\tilde{i}_{max}$  be the the maximum displacement vector in  $S_t^k$ . Then, we retrieve the normalized  $\tilde{S}_{t-1}^k = \left\{ \left( \frac{\tilde{i}}{w_s} \cdot \tilde{i}_{max}, \tilde{j} \right) | (\tilde{i}, \tilde{j}) \in S_{t-1}^k \right\}$ . Finally, we warp  $a_{t-1}^k$  to its corresponding representation to

$$\hat{a}_{t-1}^k(i, j) = a_{t-1}^k(i + \tilde{i}, j + \tilde{j}) \quad (4.3.4)$$

at the current time step  $t$ . In contrast to our baseline, we perform now a linear operation between warped  $\hat{a}_{t-1}^k$  and the present activation map  $a_t^k$  as demonstrated below:

$$f(\hat{a}_{t-1}^k, a_t^k) = \hat{a}_{t-1}^k \cdot w_1^k + a_t^k \cdot w_2^k. \quad (4.3.5)$$

This approach has the advantage that it considers the changed feature arrangement and hence the transformed map  $\hat{a}_{t-1}^k$  should be more similar to the present activation map  $a_t^k$ .

## 4.4 Training and Parameter Optimization

This section describes the loss function as well as the training and optimization procedure of *SqueezeSeg*. Afterward, we present our utilized parameters.

### 4.4.1 Loss Function

The last *Conv14* layer describes a function

$$F: \mathbb{R}^n \times \mathbb{R}^{h \times w \times c} \rightarrow \mathbb{R}^{h \times w \times |C_{ls}|}, \quad (4.4.1)$$

which maps the  $n$ -th dimensional parameter vector of *SqueezeSeg*  $\theta_{t-1} \in \mathbb{R}^n$  at time  $t - 1$  and the current input  $OSP_t$  to an unscaled activation matrix  $A$ . This representation consists of the same horizontal and vertical resolution as the original  $OSP_t$  input. The set

$$C_{ls} = \{0, 1, 2, 3\} \quad (4.4.2)$$

represents our primary classes *don't care*, *car*, *pedestrian* and *cyclist* respectively. Each scalar  $a_{ijl}$  in  $A$  represents the corresponding strength of convenience at position  $(i, j)$  that the original input channel  $C(i, j)$  belongs to the label  $l \in C_{ls}$ . However, these values show in most cases an unnormalized representation since the sum for all activation scalars is unequal one with respect to an arbitrary scalar  $a_{ijl}$ . This pipeline applies the following *softmax* function in order to retrieve the normalized probability distribution of the point-wise activation:

$$\begin{aligned} \sigma: \mathbb{R}^{|C_{ls}|} &\rightarrow [0, 1]^{|C_{ls}|} \\ \sigma(a_{ijl}) &= \frac{e^{a_{ijl}}}{\sum_{z \in C_{ls}} e^{a_{ijz}}}. \end{aligned} \quad (4.4.3)$$

The *softmax* function squashes all activation scalars  $a_{ijl}$  between zero and one.  $gt(i, j) \in C_{ls}$  denotes the corresponding ground truth label  $l$  of the input pixel  $C(i, j)$ . Afterward, the pipeline applies the cross-entropy function

$$J_{ij}: C_{l_s} \times \mathbb{R}^{|C_{l_s}|} \rightarrow \mathbb{R}$$

$$J_{ij}(gt(i, j), \sigma(a_{ij})) = b_{mask}(i, j) \cdot w_{loss}(i, j) \cdot \left( - \sum_{l \in C_{l_s}} gt(i, j) \log(\sigma(a_{ijl})) \right) \quad (4.4.4)$$

in order to determine the error rate  $J(gt(i, j), \sigma(a_{ij}))$  for each estimated pixel probability  $\sigma(a_{ijl})$ . The loss function scales the error through the multiplication with a constant loss weight map  $w_{loss} \in \mathbb{R}^{h \times w}$ . The purpose of this additional variable is to balance incorrect classifications with respect to the label distribution. As earlier mentioned, both datasets show unequal label distributions and hence the weight  $w_{loss}(i, j)$  scales the prediction errors by higher loss weights when the corresponding ground truth value  $gt(i, j)$  belongs to a rare representing label like *cyclist* or *pedestrian*. Instead, the corresponding loss weight of the *car* and especially of *don't care* represents a much shorter penalty weight due to their higher occurrence. Table 4.5 also summarizes all class weights used during training. The boolean filter

$$b_{mask}(i, j) \begin{cases} 1, & \text{if channel } C(i, j) \text{ is not empty} \\ 0, & \text{else} \end{cases} \quad (4.4.5)$$

sets the error function  $J_{ij}$  for a specific pixel  $(i, j)$  to zero, when the corresponding input channel  $C(i, j)$  references to an empty channel. In this case, the whole loss function  $J_{ij}(gt(i, j), \sigma(a_{i,j}))$  drops to zero through the zero scalar multiplication. This allows us to restrict the loss function only on relevant input information. Finally, the total loss used within parameter optimization is:

$$J(A) = \frac{\sum_{i \in H} \sum_{j \in W} J_{ij}(gt(i, j), \sigma(a_{ij}))}{\kappa \cdot \sum_{i \in H} \sum_{j \in W} b_{mask}(i, j)}. \quad (4.4.6)$$

The purpose of the division term is to determine the average error by considering the total number of valid entries. Moreover, the constant  $\kappa \in \mathbb{R}_{>0}$  scales the denominator to avoid fast divergence of the loss function  $J$ .

#### 4.4.2 Momentum Optimization

All extensions employ the momentum optimizer from tensorflow to optimize the weights and bias parameters. This optimizer is a small modification on the common stochastic gradient descent (SGD). SGD modifies the weights gradually by moving their values in the opposite gradient direction such that the overall loss is minimized. However, the main limitation of the SGD optimizer is its oscillat-

ing behavior for poorly conditioned problems where the gradient is close to being orthogonal to the shortest descend direction as illustrated in [24]. Instead, the momentum optimizer counteracts this problem by smoothing the descent directions. The idea stems from the physical equivalence of a ball rolling down a hill. The ball accelerates on steep slopes and its momentum increases. If the momentum is large enough the ball will continue to go in a certain direction independent of small hills in its way. As the gradient of the slope decreases, the ball becomes slower and slower until it reaches its local deepest valley. This behavior is equivalent according to the momentum parameter optimization:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \nabla J(F(\theta_{t-1}, OSP_t)), \\ \theta_t &= \theta_{t-1} - \mu_t \cdot v_t. \end{aligned} \tag{4.4.7}$$

The current accumulation  $v_t \in \mathbb{R}_{>0}$  at time  $t$  depends on the momentum  $\gamma \in \mathbb{R}_{>0}$ , the chain link of prior accumulations  $v_{t-1}$  and the direction of the gradient  $\nabla J(F(\theta_{t-1}, OSP_t))$ . The momentum term decreases for the dimensions whose gradient point in different directions and otherwise increase weight updates  $\theta_t$ . As a result, the momentum optimizer tends converges faster in contrast to standard SGD optimizer. The momentum hyperparameter  $\gamma$  is intended to be set between  $0 < \gamma < 1$ . In our case, we choose the initial value of  $\gamma = 0.9$ . Normally, the momentum optimizer utilizes a specific number  $b \in \mathbb{N}_{>1}$  of training examples for the calculation of one weight-update. To accomplish that, it uses the mini batch momentum optimization as follows:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \sum_{i=1}^b \nabla J(F(\theta_{t-1}, OSP_i)), \\ \theta_t &= \theta_{t-1} - \mu_t \cdot v_t. \end{aligned} \tag{4.4.8}$$

Besides, the pipeline reduces the learning rate of  $\mu_t \in \mathbb{R}_{>0}$  with an exponential decay method to ensure a stable convergence:

$$\mu_t = \mu_{t-\rho} \cdot \lambda^{\frac{t}{\rho}}. \tag{4.4.9}$$

$t \in \mathbb{N}_{>0}$  describes the global training step, and  $\rho \in \mathbb{N}_{>0}$  denotes a specific time step where the learning rate decays. The scalar multiplication between the decay factor  $\lambda^{\frac{t}{\rho}}$  and prior learning rate  $\mu_{t-\rho}$  updates the present learning rate  $\mu_t$ .

### 4.4.3 Training Environment and Evaluation Metrics

We base our work on *SqueezeSeg* [30] and developed all further experimental extensions in the widespread open source framework [1] tensorflow. Furthermore, we trained and evaluated all models on a Nvidia GeForce GTX Titan equipped with 6GB graphics memory. To determine the segmentation performance for all experiments, we utilize three different metrics: Precision (Pre), recall (Rec) and intersection over union (IoU) as illustrated below:

$$Pre_l = \frac{|P_l \cap G_l|}{|P_l|}, \quad (4.4.10)$$

$$Rec_l = \frac{|P_l \cap G_l|}{|G_l|}, \quad (4.4.11)$$

$$IoU_l = \frac{|P_l \cap G_l|}{|P_l \cup G_l|}. \quad (4.4.12)$$

$P_l \in \{0, 1\}^{h \times w}$  denotes the pixel-wise prediction of *SqueezeSeg* and  $G_l \in \{0, 1\}^{h \times w}$  the corresponding ground truth belonging to one of our primary classes  $l$ .  $Pre_l$  describes the ratio between the relevant instances among the predicted label set  $P_l$ . This value yields a high number for a specific class  $l$  if the prediction set  $P_l$  of *SqueezeSeg* includes many overlapping samples in the ground truth set  $|G_l|$  and the amount of predicted labels is  $|P_l| > 1$ . Instead, the recall or otherwise known as the sensitivity describes the fraction between  $|P_l \cap G_l|$  and the cardinality of ground truth set  $G_l$ . The intersection over union defines the ratio between the number overlapping samples in  $|P_l \cap G_l|$  among the union of the predicted label set  $P_l$  and ground truth set  $G_l$ .

### 4.4.4 Selection of the Hyperparameters

The following Table 4.5 shows the initial hyperparameter configuration which we determined empirically by several training procedures. In total, we used two different training strategies: The training using the *concatenation* extension as well as the combination of activation representations, assume that the prior and the current frame are successive. Hence, we deemed it necessary for the first training method to organize the whole training batch in a sequential order which includes at any time exactly one specific scene. However, the loss function made hesitantly progress to the local minimum during the parameter optimization. Moreover, the trained model overfitted quickly by considering only the frequent categories *car* and *don't care*. Simultaneously, the metric results of *pedestrian* and *cyclist* kept on a constant zero value, even after 100000 training iterations.

#### 4.4 Training and Parameter Optimization

Hyperparameters	Symbols	Random Batch	Ordered Batch
batch size	$b$	10	10
learning rate	$\mu$	0.01	0.001
decay rate	$\lambda$	0.75	0.75
decay steps	$\rho$	10000	100000
momentum	$\gamma$	0.9	0.9
loss coefficient	$\kappa$	15	15
loss weight	$w_{don't\ care}$	0.0067	0.0067
	$w_{car}$	1	1
	$w_{cyc}$	10	12
	$w_{ped}$	10	12

Table 4.5: Overview of utilized hyperparameters.

We increased the loss weights  $w_{cyc}$  and  $w_{ped}$ , trying to compensate the overfitting behavior on *car*. This measure did not constitute an aid because the IoU score of the category *car* dropped nearly to zero and the network tended to estimate most of the pixels as *cyclist* and *pedestrian*. Building upon these results, we found it less reasonable to perform further hyperparameter optimizations on the ordered batch. Subsequently, we changed our training strategy and randomized the batch by choosing frames from arbitrary scenes. As a result, the loss function for the *single frame* mechanism made rapidly progress to the local minimum and the IoU training results of the classes *pedestrian*, as well as *cyclist*, always kept far above of the zero threshold.

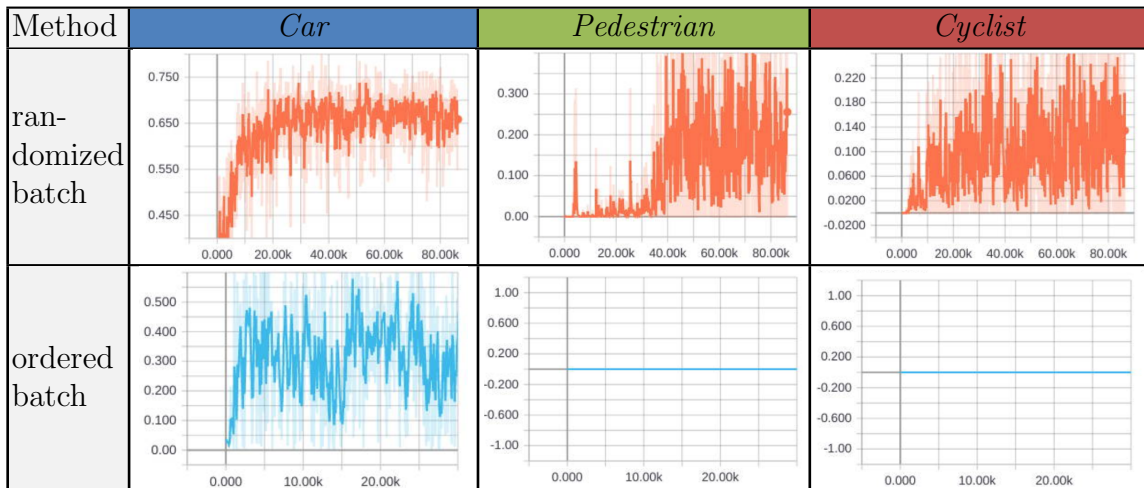


Table 4.6: **Training comparison:** Randomized batches exhibit significant better IoU than ordered batches.

In order to ensure that our extensions combined only successive frames and activations, we chose an arbitrary pair of two adjacent frames to create a randomized batch of size  $2b$ . The Table 4.6 illustrates the considerable impact of using a randomized batch (upper row) in contrast to an ordered batch (lower row). The vertical axis describes the IoU score and the horizontal axis denotes the corresponding training iteration step. These IoU metric scores base on the training set split as illustrated in Table 4.2.

## 4.5 Point Cloud Normalization

The LIDAR input data varies from one frame to another considerably. Unnormalized input vectors lead in general to over- and undercompensation for different weight parameters and reduce the networks generalization capabilities. Furthermore, the employed loss function tends to diverge rapidly without any input normalization. First, we determine the number of valid channels by

$$k = \sum_{f \in F} \sum_{i \in H} \sum_{j \in W} b_{mask}(i, j). \quad (4.5.1)$$

The total number of input channels depends on the *OSP* frame set  $F$  with respective to the utilized dataset, the height  $h$  and width resolution  $w$ , respectively. Building upon these results, we calculate the mean for all valid entries

$$mean_c = \frac{1}{k} \sum_{f \in F} \sum_{i \in H} \sum_{j \in W} C_f(i, j) \cdot b_{mask}(i, j) \quad (4.5.2)$$

as well as the empirical standard deviation

$$std_c = \frac{1}{k} \sqrt{\sum_{f \in F} \sum_{i \in H} \sum_{j \in W} (mean_c - C_f(i, j))^2 \cdot b_{mask}(i, j)} \quad (4.5.3)$$

for each feature vector considering all channels  $C_f(i, j)$  stored in a specific frame  $f$ . The pipeline scales each input channel  $C(i, j)$  before the training process by

$$C(\tilde{i}, \tilde{j}) = \frac{C(i, j) - mean_c}{std_c}. \quad (4.5.4)$$

Subsequently, the mean of the input data is reduced to zero and the division by the standard deviation scales the channels standard deviation to one.



# 5 Evaluation

The results of our experiments base on the evaluation of two separate datasets which include different FoV annotations. In the case of the 81° FoV annotated dataset from Geiger et al. [10], we retrieve the ground truth information by utilizing the OBB in the corresponding image frames. In contrast, the dataset from Behley et al. [3] consists of a point-wise full FoV annotation. Another significant difference is that the resulting *OSP* frames from the 360° FoV include in average more empty entries. For a more detailed description of the employed dataset, we refer the reader to the previous Section 4.1. In the following, we analyze the total number of parameters and the corresponding runtime with respect to each extension. Section 5.2 presents the segmentation results as well as some representative pixel-wise prediction plots. We hypothesized that each extension provides a benefit to the standard *single frame* mechanism. Hence, the last Section 5.3 highlights the impact for each extension and outline if the experiments performed as expected.

## 5.1 Parameter and Runtime Analysis

The purpose of this section is to analyze the total number of parameters and the corresponding runtime of each method.

Figure 5.1 outlines that the total number of parameters is independent of the height and width dimension of the *OSP* input. As we already described in Section 3.3, the number of parameters depends on the number of filter  $f_i$  times the dimension of each kernel. Let denote  $k_{i_w}$  the kernel width and  $k_{i_h}$  the kernel height to the corresponding *layer<sub>i</sub>* located at the  $i$ -th position in the *SqueezeSeg* architecture. Then, we calculate the total number of parameters as

$$Param = \sum_{i=0}^n (k_{i_w} k_{i_h} c + 1) f_i, \quad (5.1.1)$$

where we increase the product of the inner brackets by one, since the equation has additionally to consider the bias value for each kernel  $k_i$ . The extensions *single frame* and *augmentation* demonstrate the lowest usage of employed parameters.

## 5 Evaluation

These methods use on both resolution sizes a constant input channel size of  $c = 5$ .

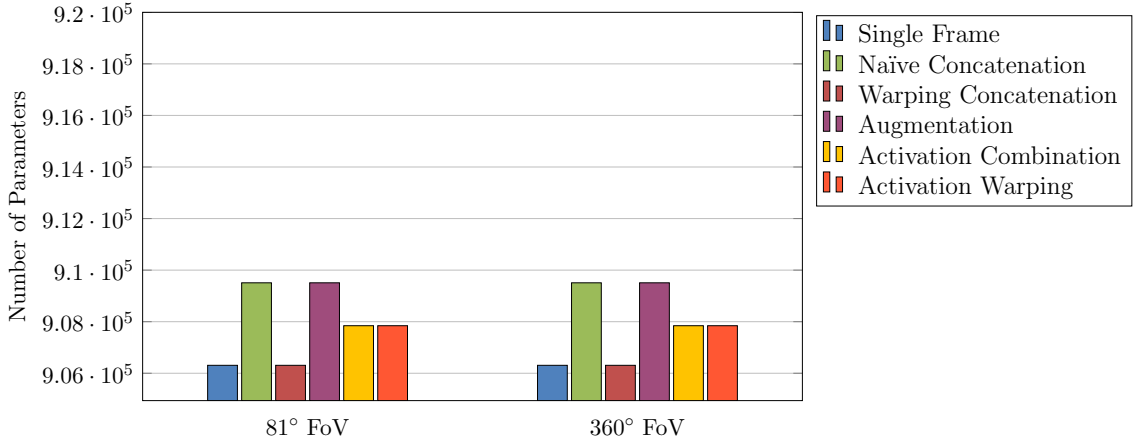


Figure 5.1: **Evaluated dataset comparison:** Magnitude of applied parameters for each extension.

Moreover, the *naïve-* as well as the *warp concatenation* extensions employ twice as many of input features  $c = 10$ . This method leads to a larger amount of parameters regarding the parallel convolution layer *Conv1a* and *Conv1b* at the beginning of the network structure. As a result, both methods take 3000 more parameters compared to the *single frame* mechanism. Instead, the *activation combination* and *activation warping* methods take advantage of additional 1536 channel weights beside the total number of layer parameters. However, the resolution size has a considerable impact on the runtime, since the number of kernel operations increases by resolution sizes. Hence, our two different datasets show noticeable differences concerning input complexity as well as runtime. Our calculated *OSPs* based on the 81° FoV yield a resolution size of  $64 \cdot 384 \cdot c$ . Instead, the frames using the full FoV consist of size  $64 \cdot 1920 \cdot c$ . Figure 5.2 demonstrates that in average the usage of 81° FoV frames yields a significant lower runtime compared to the full FoV frames. Higher input resolution also causes larger activation maps. However, this has a crucial impact on the computational effort of the following layer operations. The *single frame* approach yields on both resolutions the lowest runtime since it uses the smallest amount of parameters and is independent of preprocessing tasks. Processing of a full FoV frame takes approximately 3.4 ms and meets the runtime demands of point cloud sequences of  $\sim 30$  Hz.

The *naïve concatenation* as well as the *warp concatenation* employs 3000 more parameters compared to the *single frame* mechanism. Furthermore, the *warp concatenation* increases the number of computations by warping the prior input into the current representation. Both input concatenation approximately double

their runtime when increasing the FoV from 81° to 360°.

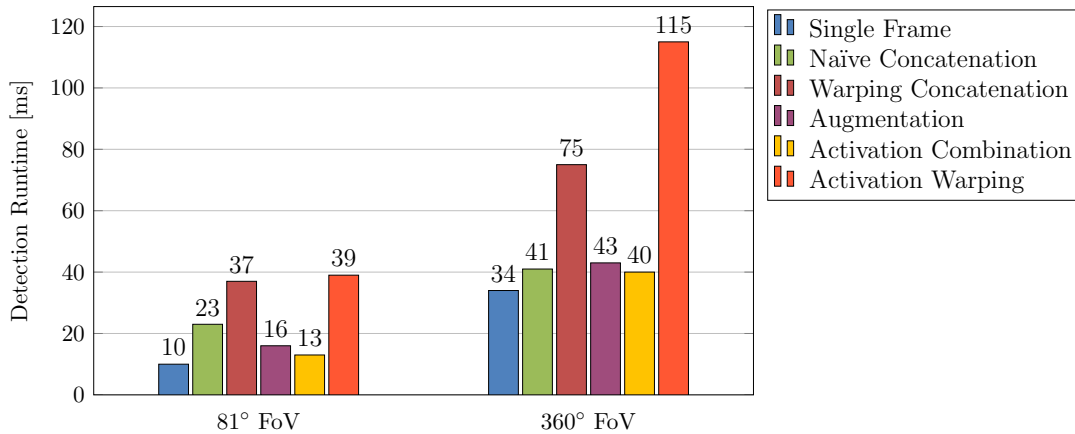


Figure 5.2: **Evaluated dataset comparison:** Detection runtime for each extension.

The *augmentation* of LIDAR point clouds aims to complete empty *OSP* channels by transforming  $n$  prior LIDAR clouds into the current representation through ego-motion. This operation results for  $n = 1$  in a small computational effort which is characterized by additional 6 or 9 ms, respectively. Higher usage of  $n$  prior point clouds also causes a linear increase in computational cost. In contrast, the *activation combination* and *activation warping* fuse three specific activation maps of the desired layer and both of them apply in overall additional 1536 channel weights. The fuse operation consists of a linear combination of prior and current activation as illustrated in Figure 4.9. Moreover, the *activation warping* method demonstrates the highest number of computations. This extension retrieves the 2D index displacement between two adjacent frames and subsequently exploits this information to warp the prior activation map into the representation of the current activation. The warping concept needs large channel rearrangement operations, and hence the usage of larger activation resolutions increase the computational effort by a considerable factor.

## 5.2 Experimental Metric Results

The following Table 5.1 and Table 5.2 show the segmentation results for all experiments considering the primary classes *car*, *pedestrian* and *cyclist*. Table 5.1 demonstrates the metric scores of the 81° FoV dataset, whereas Table 5.2 represents the results of the 360° LIDAR dataset. Our evaluation shows that *SqueezeSeg* performs much better on the class *car* compared to *pedestrian* and *cyclist*. Upon closer observation, there is an apparent correlation between the metric results and

## 5 Evaluation

the pixel-wise label distribution in Table 4.3. *Pedestrians* and *cyclists* reflect a much smaller amount of LIDAR points and also share considerable more complex geometrical features in comparison to a *car*. Moreover, there is also a strong correlation between a *pedestrian* and a *cyclist* riding the bike with respect to the geometrical features. Hence, it is more challenging to learn the distinction between a *cyclist* and *pedestrian* than between a vehicle and a human.

### Segmentation Results Using 64 Vertical Beams

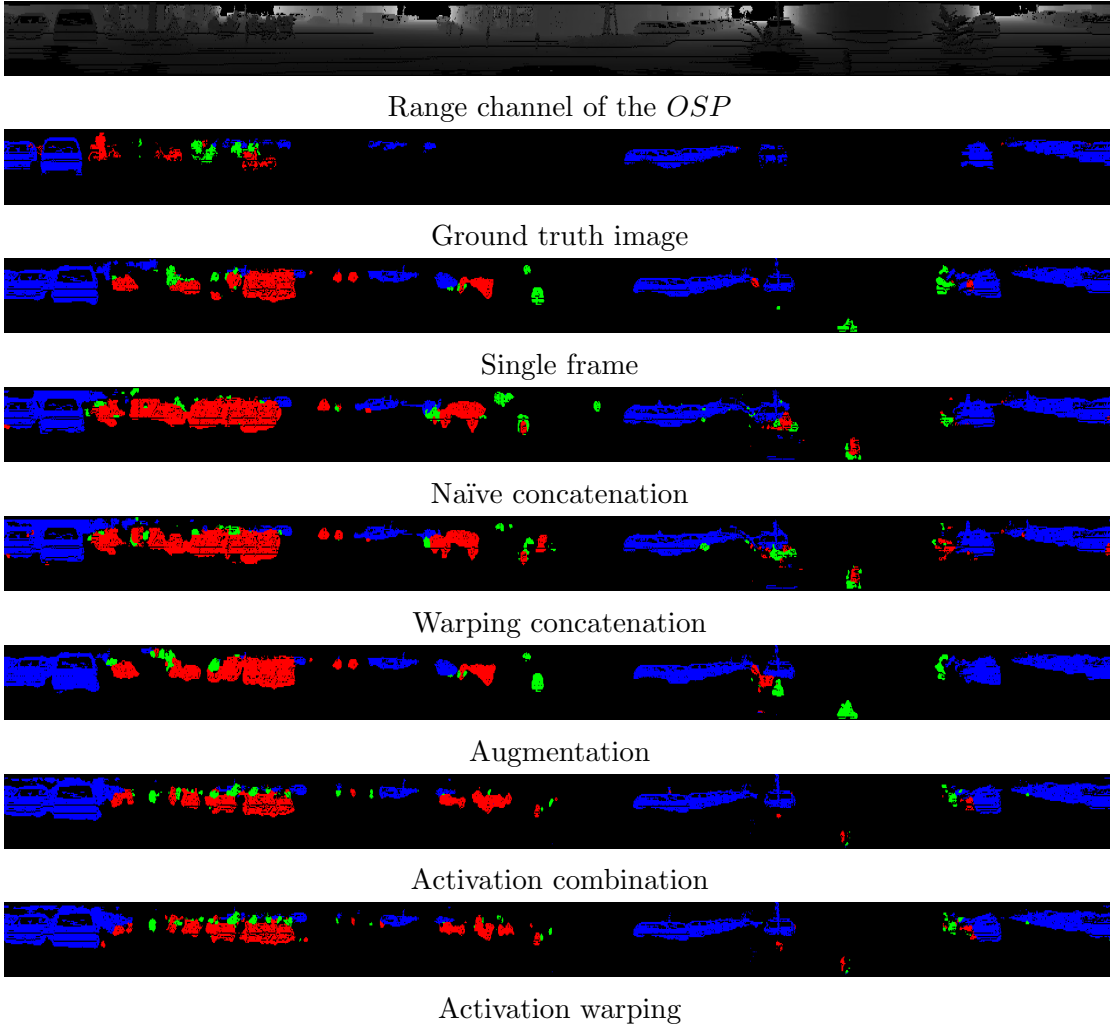
Class	Car			Pedestrian			Cyclist			Avg.
	Prec.	Rec.	IoU	Prec.	Rec.	IoU	Prec.	Rec.	IoU	IoU
Single Frame	<b>0.591</b>	<b>0.950</b>	<b>0.573</b>	0.087	0.132	0.055	<b>0.151</b>	0.342	<b>0.117</b>	<b>0.248</b>
Naïve Concatenation	0.555	0.942	0.537	<b>0.089</b>	<b>0.258</b>	<b>0.071</b>	0.071	0.389	0.064	0.224
Warp. Concatenation	0.374	0.863	0.353	0.018	0.130	0.016	0.004	0.016	0.03	0.133
Augmentation	0.577	0.889	0.538	0.063	0.195	0.050	0.077	<b>0.499</b>	0.070	0.219
Activation Comb.	0.321	0.900	0.309	0.025	0.056	0.018	0.008	0.053	0.007	0.113
Activation Warping	0.319	0.899	0.308	0.026	0.057	0.018	0.008	0.050	0.007	0.111

Table 5.1: **Segmentation results:** KITTI’s 81° FoV Dataset.

Class	Car			Pedestrian			Cyclist			Avg.
	Prec.	Rec.	IoU	Prec.	Rec.	IoU	Prec.	Rec.	IoU	IoU
Single Frame	0.716	<b>0.961</b>	<b>0.696</b>	<b>0.044</b>	0.207	<b>0.038</b>	<b>0.178</b>	0.676	<b>0.164</b>	<b>0.299</b>
Naïve Concatenation	0.645	0.945	0.624	0.027	<b>0.217</b>	0.025	0.117	0.665	0.111	0.252
Warp. Concatenation	0.648	0.940	0.622	0.020	0.196	0.019	0.104	0.680	0.099	0.246
Augmentation	<b>0.721</b>	0.933	0.686	0.039	0.203	0.034	0.135	<b>0.703</b>	0.128	0.282
Activation Comb.	0.654	0.952	0.633	0.033	0.186	0.029	0.118	0.552	0.107	0.256
Activation Warping	0.654	0.954	0.634	0.034	0.190	0.030	0.118	0.551	0.108	0.257

Table 5.2: **Segmentation results:** 360° FoV Dataset.

The following results show that the experiments perform better on the full FoV compared to the insufficient 81° FoV. A possible explanation is that the dataset of Behley et al. [3] includes precise object annotations. Instead, the oriented bounding boxes exhibit the problem that it also covers unnatural parts like ground which are no part of the classes *car*, *pedestrian* or *cyclist*. The object annotations do not represent in every frame constantly mislabeled objects as ground and hence *SqueezeSeg* might learn misinterpreted patterns. The full FoV *OSP* representation complete objects at the borders which is not the case with the truncated image. The following Figure 5.3 illustrates the corresponding pixel-wise predictions of each extension, using the 360° FoV dataset.

Figure 5.3: **Segmentation results:** 360° FoV Dataset.

### Micro Aerial Vehicle Simulation Using 16 Vertical Beams

Nowadays LIDAR sensors suitable for MAVs explore also the full FoV with the restriction of 16 vertical beams. The provided 3D LIDAR point cloud of such a sensor consists of four times less information about the environment in contrast to a common 64 beam vertical LIDAR scanner. As we already mentioned before, our utilized datasets base on the measurements of the Velodyne HDL-64E. In order to simulate the performance on a 16 vertical sized point cloud, we take every 4-th vertical beam and store it in a new *OSP* frame. The following evaluation restricts only on the *single frame*, *naïve concatenation*, *augmentation* and *activation combination* extensions, since we find it less reasonable to evaluate the *warped* extensions which provide no real benefit on more valuable input representations.

## 5 Evaluation

The parameters which were optimized on an *OSP* with a vertical size of 64, perform on the *OSP* with 16 vertical beam scan poorly. Hence, we refined the model parameters with respect to each extension. The following segmentation results in Table 5.3 base on the 360° FoV dataset from Behley et al. [3]:

Class	Car			Pedestrian			Cyclist			Avg.
	Prec.	Rec.	IoU	Prec.	Rec.	IoU	Prec.	Rec.	IoU	
Single Frame	0.493	0.919	0.472	0.016	0.160	0.014	0.044	<b>0.511</b>	0.042	0.176
Naïve Concatenation	0.393	0.922	0.380	0.008	0.155	0.007	<b>0.053</b>	0.190	0.043	0.143
Augmentation	0.511	0.941	0.495	0.015	<b>0.209</b>	0.014	0.009	0.044	0.008	0.172
Activation Comb.	<b>0.550</b>	<b>0.944</b>	<b>0.532</b>	<b>0.026</b>	0.095	<b>0.021</b>	0.039	0.342	0.036	<b>0.196</b>

Table 5.3: **Segmentation results:** 360° FoV Dataset using 16 vertical beams.

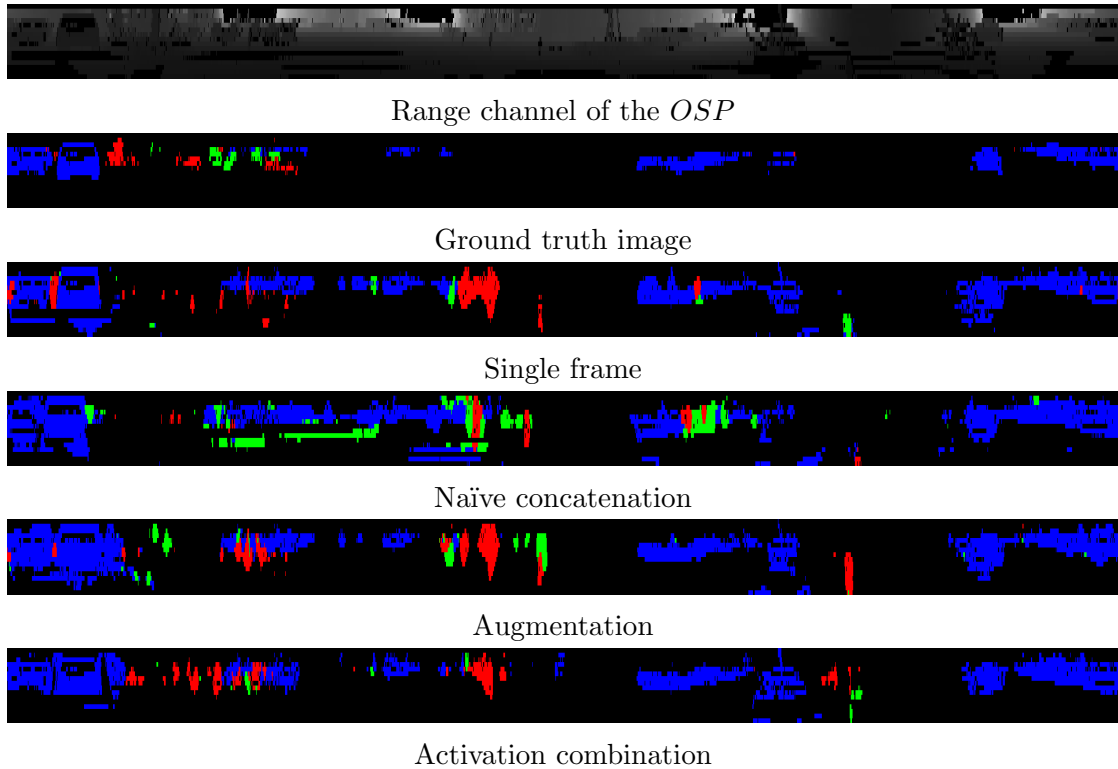


Figure 5.4: **Segmentation results:** 360° FoV Dataset using 16 vertical beams.

## 5.3 Experimental Results Analysis

This last section highlights the impact for each extension and outline if the experiments performed as expected.

### 5.3.1 Extension 1.1: Naïve Concatenation

This extension aims to combine the raw data of two successive frames to provide *SqueezeSeg* with more input information. In particular, this method assumes that the environment does not change considerably since it ignores the ego-motion during the time differences  $t$  and  $t - 1$ . However, the corresponding results in Table 5.1, Table 5.2 and Table 5.3 show that this extension underperforms in average the single frame mechanism and hence demonstrate an unanticipated finding. This result may be explained by the fact that concatenated raw input data represent in most cases larger discrepancy as expected. This would have a considerable impact on the feature extraction and hence explain the worse outcome. We can observe a slight improvement in segmenting *pedestrians* according to Table 5.1. Besides, this experiment increases also the number of parameters and the runtime in comparison to the single frame detection. Building upon these results, this extension does not demonstrate in average a positive impact.

### 5.3.2 Extension 1.2: Warping Concatenation

In comparison to the *naïve concatenation*, the *warping concatenation* extension demonstrates a better correlation between previous and current frame by utilizing the ego-motion estimate. Hence, we expected that this extension shows additional benefit according to the *naïve concatenation*. Contrary to expectations, the evaluation results do not pose a significant benefit to warp the previous *OSP* into the current representation. The evaluation clearly outlines that this operation performs in the average worse compared to the first extension. Urban environments usually consist of individual dynamic moving objects. The discrepancy between the *naïve* and *warping concatenation* could be attributed to the ego-motion estimate since it yields the limitation that it only considers the movement of the sensor and does not include the individual movement of objects in the surrounding area. Another possible explanation is that the warped frame includes in the average more empty entries in comparison to the standard frame since the warping operation sometimes moves several points from the prior frame to one position  $(i, j)$  in the current representation. As a result, only one point can be stored at position  $(i, j)$  and other areas of the image are empty. Despite its weaker performance, this

method also increases the runtime by 14 or 34 ms with respect to the dataset. In summary, this extension demonstrates a negative impact in view of computational effort as well as segmentation results.

### 5.3.3 Extension 2: Augmenting Current LIDAR Input Using Ego-motion Estimates

The primary motivation of this extension is to reuse LIDAR features of prior frames to increase the input density of the current  $OSP$ . This extension transforms  $n$  point clouds with the ego-motion estimate in a reference view and subsequently complete as many entries as possible in the current input. A higher preset of  $n$  causes a linear increase of 6 or 9 ms respectively and does not provide a better segmentation result. A possible explanation is that two successive point clouds are strongly correlated. It is quite likely that  $n > 1$  previous point clouds contain objects which are partially occluded in the current  $OSP_t$  frame. Hence, the ARV module transforms more points of objects currently hidden in the reference view since this method always prefers points which are closest to the sensor. However, the module completes with only one prior point cloud on average 16.29% of the empty entries in current input  $OSP_t$ . In particular, the estimated cars (blue) on the left side in Figure 5.3, demonstrate that this method provides pixel-wise predictions of objects which represent much less empty entries compared to the other extensions. The metric results in Table 5.2 and Table 5.3 show that this extension provides a slight benefit in cases of cars when using 16 instead of 64 vertical beams. The reason for this is not apparent, but we assume that an  $OSP$  with four times less data gains more information about the environment from a single completed entry. In comparison to the previous extensions, this method does not increase the input complexity and hence shows lower detection runtime. It should be noted that the augmentation is limited to the accuracy of the ego-motion estimates and hence the segmentation results can vary by each dataset. In conclusion, this strategy meets our expectations partially since it provides an advantage for the full FoV and also just in cases of *cars*.

### 5.3.4 Extension 3: Activation Combination and Activation Warping

Previous approaches like Gadde, Jampani, and Gehler [9] provide a strategy which benefits from the re-usage of prior activations in IS tasks. The purpose of our extension is to combine prior and current activations of three different layers. However, we restrict this extension on the *Fire3*, *Fire8* and *FireDeconv11* layers which



output activation maps on different scales. In contrast to the *concatenation* experiment, this method aims to exploit the prior activation and to combine it with the current internal representation through a weighting function. The function weights each activation channel with a trainable parameter of the successive representations in order to restrict on significant features. According to Table 5.2, this strategy underperforms the *single frame* mechanism and outperforms both *concatenation* extensions when using 64 vertical beams. An implication of this is the possibility that the weighting function is rather appropriate to combine prior and current features than to concatenate raw input features. Fortunately, this method outperforms in average the *single frame* with respect to the 16 vertical sized *OSP* in cases of *car* and *pedestrian* as illustrated in Table 5.2. Furthermore, we also implemented an extension which utilizes the ego-motion estimate additionally to perform the warping operation on the internal representations. Surprisingly, there is no considerable difference with respect to the segmentation results between the *activation warping* and *activation combination*. We assume that this warping method suffers from the same disadvantages as highlighted in the *naïve concatenation*. In view of computational efficiency, the warping concept increases the runtime tremendously and hence does not provide any additional positive impact.



## 6 Conclusion

The primary goal of this thesis was to examine the impact of exploiting ego-motion estimates and previous LIDAR scans on LIDAR segmentation. However, this thesis restricts to the organized point cloud representation and employs a convolutional neural network which aims primarily at computational efficiency.

Despite its exploratory nature, this work evaluated experiments on  $81^\circ$  and also on the complete  $360^\circ$  field of view dataset annotations.

Experimental results show that the presented concatenation and warping methods of prior frames or network activations underperform in contrast to common single frame mechanisms. Instead, transforming previous input representations into the current point cloud yields a considerable benefit to complete insufficient input frames and demonstrates the feasibility to enhance segmentation results.

Another major finding was that the combination of internal activation representations through a weighting function demonstrates a strategy to improve results when using LIDAR scans with smaller vertical resolution.

More broadly, research is also needed to investigate the influence of prior point clouds depending on a large-scaled voxel representation which show more potential to augment insufficient frames as well as studying 3D spatial complexes. Another possible area of future research would be to examine warp methods which utilize 3D LIDAR-based Scene- or optical flow to consider the dynamic movement of the surrounding area.



# Bibliography

- [1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. “Tensorflow: a system for large-scale machine learning.” In: *OSDI*. Vol. 16. 2016, pp. 265–283.
- [2] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Mahmudul Hasan, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. “The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches”. In: *arXiv preprint arXiv:1803.01164* (2018).
- [3] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jürgen Gall. “A Dataset for Semantic Segmentation of Point Cloud Sequences”. In: submitted to CVPR. 2019.
- [4] Jens Behley and Cyrill Stachniss. “Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments”. In: *Proc. of Robotics: Science and Systems (RSS)*. 2018.
- [5] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. “High accuracy optical flow estimation based on a theory for warping”. In: *European conference on computer vision*. Springer. 2004, pp. 25–36.
- [6] Ayush Dewan, Tim Caselitz, Gian Diego Tipaldi, and Wolfram Burgard. “Rigid scene flow for 3D LiDAR scans”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), pp. 1765–1770.
- [7] Ayush Dewan, Gabriel L Oliveira, and Wolfram Burgard. “Deep semantic classification for 3d Lidar data”. In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE. 2017, pp. 3544–3549.
- [8] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. “Flownet: Learning optical flow with convolutional networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2758–2766.

## Bibliography

- [9] Raghudeep Gadde, Varun Jampani, and Peter V Gehler. “Semantic video cnns through representation warping”. In: *CoRR*, abs/1708.03088 8 (2017), p. 9.
- [10] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. “Vision meets Robotics: The KITTI Dataset”. In: *International Journal of Robotics Research (IJRR)* (2013).
- [11] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.
- [12] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan Dirk Wegner, Konrad Schindler, and Marc Pollefeys. “Semantic3D.net: A new Large-scale Point Cloud Classification Benchmark”. In: *CoRR* abs/1704.03847 (2017). arXiv: 1704.03847. URL: <http://arxiv.org/abs/1704.03847>.
- [13] David J Heeger and Allan D Jepson. “Subspace methods for recovering rigid motion I: Algorithm and implementation”. In: *International Journal of Computer Vision* 7.2 (1992), pp. 95–117.
- [14] Evan Herbst, Xiaofeng Ren, and Dieter Fox. “Rgb-d flow: Dense 3-d motion estimation using color and depth”. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 2276–2282.
- [15] Berthold KP Horn and Brian G Schunck. “Determining optical flow”. In: *Artificial intelligence* 17.1-3 (1981), pp. 185–203.
- [16] Forrest Iandola, Song Han, Matthew Moskewicz, Khalid Ashraf, William Dally, and Kurt Keutzer. “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size”. In: *arXiv preprint arXiv:1602.07360* (2016).
- [17] Joseph K Kearney, William B Thompson, and Daniel L Boley. “Optical flow estimation: An error analysis of gradient-based methods with local optimization”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2 (1987), pp. 229–244.
- [18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.
- [19] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Lake Waslander. “Joint 3D Proposal Generation and Object Detection from View Aggregation”. In: *CoRR* abs/1712.02294 (2017). arXiv: 1712.02294. URL: <http://arxiv.org/abs/1712.02294>.

- [20] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. “Deep Continuous Fusion for Multi-Sensor 3D Object Detection”. In: *ECCV*. 2018.
- [21] Lingni Ma, Jörg Stückler, Christian Kerl, and Daniel Cremers. “Multi-view deep learning for consistent semantic mapping with rgb-d cameras”. In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE. 2017, pp. 598–605.
- [22] Moritz Menze and Andreas Geiger. “Object scene flow for autonomous vehicles”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3061–3070.
- [23] F. Moosmann, O. Pink, and C. Stiller. “Segmentation of 3D lidar data in non-flat urban environments using a local convexity criterion”. In: *2009 IEEE Intelligent Vehicles Symposium*. 2009, pp. 215–220. DOI: 10 . 1109 / IVS . 2009 . 5164280.
- [24] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).
- [25] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. “Complex-YOLO: Real-time 3D Object Detection on Point Clouds”. In: *CoRR* abs/1803.06199 (2018). arXiv: 1803 . 06199. URL: <http://arxiv.org/abs/1803.06199>.
- [26] Benjamin Ummenhofer, Huizhong Zhou, Jonas Uhrig, Nikolaus Mayer, Eddy Ilg, Alexey Dosovitskiy, and Thomas Brox. “Demon: Depth and motion network for learning monocular stereo”. In: *IEEE Conference on computer vision and pattern recognition (CVPR)*. Vol. 5. 2017, p. 6.
- [27] Victor Vaquero, Alberto Sanfeliu, and Francesc Moreno-Noguer. “Deep lidar cnn to understand the dynamics of moving vehicles”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–6.
- [28] Sen Wang, Ronald Clark, Hongkai Wen, and Agathoniki Trigoni. “DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA) (2017)*, pp. 2043–2050.
- [29] Yuan Wang, Tianyue Shi, Peng Yun, Lei Tai, and Ming Liu. “PointSeg: Real-Time Semantic Segmentation Based on 3D LiDAR Point Cloud”. In: *arXiv preprint arXiv:1807.06288* (2018).

## Bibliography

- [30] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. “Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1887–1893.
- [31] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. “Conditional random fields as recurrent neural networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1529–1537.