FuseSeg: LiDAR Point Cloud Segmentation Fusing Multi-Modal Data

Georg Krispel, Michael Opitz, Georg Waltner, Horst Possegger, Horst Bischof
Institute of Computer Graphics and Vision
Graz University of Technology
{georg.krispel, michael.opitz, waltner, possegger, bischof}@icg.tugraz.at

Abstract

We introduce a simple yet effective fusion method of Li-DAR and RGB data to segment LiDAR point clouds. Utilizing the dense native range representation *of a LiDAR sensor and the setup calibration, we establish point correspondences between the two input modalities. Subsequently, we are able to warp and fuse the features from one domain into the other. Therefore, we can jointly exploit information from both data sources within one single network. To show the merit of our method, we extend SqueezeSeg, a point cloud segmentation network, with an RGB feature branch and fuse it into the original structure. Our extension called* FuseSeg *leads to an improvement of up to 18% IoU on the KITTI benchmark. In addition to the improved accuracy, we also achieve real-time performance at 50 fps, five times as fast as the recording speed of the KITTI LiDAR data.*

1. Introduction

Being able to segment objects from point clouds is crucial for driver assistant systems, autonomous cars and other robotic perception tasks. Autonomous driving requires multiple sensors to capture all relevant information of the environment. Different types of sensors compensate the individual disadvantages and ensure robust perception in challenging environments. However, fusing and leveraging all this multi-modal data is a non-trivial task.

The task of 3D perception for autonomous vehicles is usually tackled with a combination of RGB cameras and Li-DAR sensors (*i.e.* laser range scanners). Recently, numerous architectures with diverse and often complex designs for sensor fusion have been published. However, due to the complexity of this task many methods either use only single-modal input, *e.g.* [17, 37, 38] or use the benefits of multi-modalities only after single-modal proposal generation, *e.g.* [5, 25, 31]. Thus, not all available information is leveraged jointly. Objects poorly visible in one single sensor are prone to be missed.

To address this problem, we propose a simple and effec-

tive fusion method utilizing a dense native representation of laser range scanner data, such that all available information can be processed jointly by common convolutional neural network (CNN) architectures. The key idea is to warp expressive RGB features into this LiDAR representation, leveraging correspondences which can be established without any exhaustive search. In this work we focus on the task of point cloud segmentation to show the effectiveness and benefits of our fusion method.

In particular, we extend SqueezeSeg [37] with an additional branch based on MobileNetV2 [30] to leverage RGB information as well. However, naïvely warping the RGB image into range space and applying an ImageNet CNN for early fusion, *e.g.* [11] or intermediate fusion, *e.g.* [13], hampers the transfer learning benefits of CNNs, as the input image is visually distorted.

To overcome this issue, we propose to apply the ImageNet CNN on the original undistorted RGB image to better leverage the benefits of CNNs. Next, we warp the CNN features into the range space to get a dense and powerful representation. Thereby, we leverage the RGB/LiDAR calibration to establish *control points* for a polyharmonic spline interpolation [8]. We improve SqueezeSeg's segmentation results by a large margin without the use of any synthetic data (in contrast to [37, 38]).

We still perform at 50 fps on a NVIDIA GTX 1080Ti GPU, more than twice as fast as common LiDAR sensors dedicated for autonomous cars (typically operating at 20 Hz) and five times as fast as the LiDAR sensor used during the recording of the KITTI benchmark suite (10 Hz). Furthermore, we show that our approach performs better than state-of-the-art RGB semantic segmentation approaches.

2. Related Work

To better set our work in context, we will first consider recent approaches for 3D point clouds processing (Section 2.1) and then methods optimized for pseudo-3D/2.5D representations (Section 2.2). Finally, we will discuss works most related to ours, in particular about the fusion of depth and RGB information (Section 2.3).
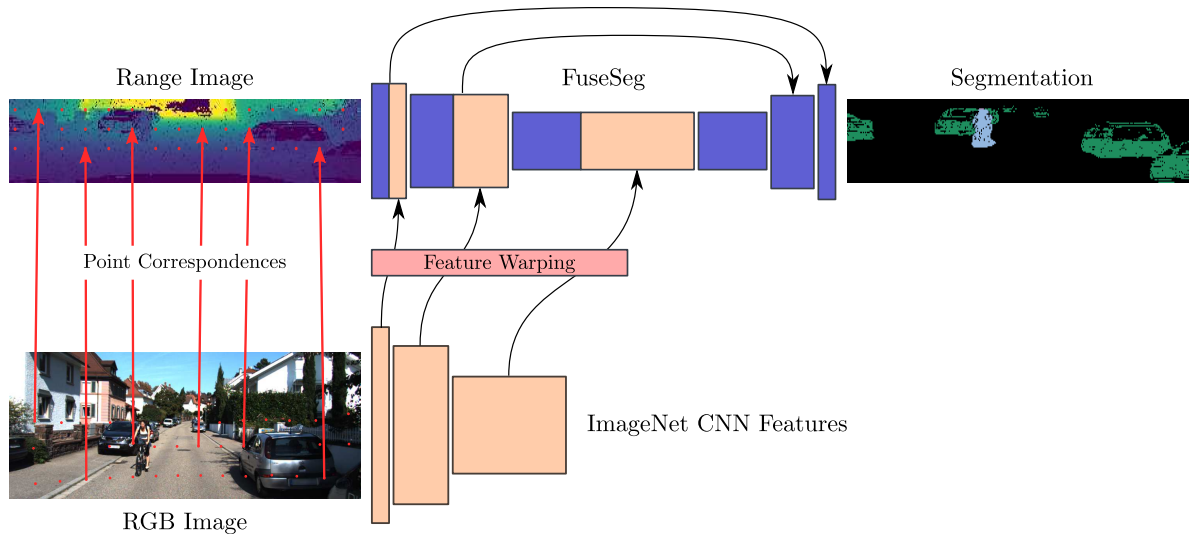
Figure 1: Schematic overview of our FuseSeg architecture. By exploiting the RGB/LiDAR calibration to establish point correspondences, we fuse feature representations from the RGB and the range image. We utilize the known correspondences to warp the RGB features such that they fit into the range image network. Our range image branch is a slightly modified SqueezeSeg [37] and we use a MobileNetV2 [30] as image branch during our experiments.

2.1. 3D Point Cloud Processing

Standard CNNs require dense input representations on uniform grids. Thus, vanilla CNNs can not be used directly on point clouds as they are sparse in 3D space. To overcome this issue, various approaches have been proposed recently. They have been applied to various tasks, *e.g.* classification, 3D object detection and (part-)segmentation. These approaches can be divided into two groups, *i.e.* direct and grid-/graph-based methods.

Direct Methods are deep architectures which are applied to the point cloud directly. One of the pioneering works in this group is PointNet by Qi *et al*. [26]. They learn multi-layer perceptrons and linear transformations to map each point individually to an expressive feature space. Subsequently, a max pooling operation generates an order-independent global feature vector, which is utilized for classification and segmentation.

PointNet lacks the ability to encode local structures with varying density. The subsequent extension PointNet++ [27] tackles this problem by introducing a hierarchical processing strategy. Multiple works [39, 19, 34] introduce a generalization of the classical convolution to irregular point sets. Same as PointNet++, they use a k-nearest neighbor search to overcome the lack of a strictly defined neighborhood.

These methods are able to process a small and fixed amount of points (up to a few thousand). To deal with larger point clouds, various strategies like tiling or farthest point sampling (FPS) must be applied to reduce the amount of processed points. Due to the varying sparsity of LiDAR point clouds, these strategies are usually not very useful

when directly applied to single sweeps, as often several samples at nearby salient regions are needed, e.g. to recover an object's outline, instead of few wide-spread samples. For example, the native choice of FPS are far distant points, which is, given a LiDAR point cloud, not valuable for any downstream task.

Grid-/Graph-based Methods apply established CNNs, transforming the point cloud into grid-based [29, 22, 33] or graph-like [36, 32] representations. The varying sparsity is the major issue here. Most of the covered space is empty and this would lead to a huge overhead by naïvely convolving over a regular 3D grid. To enable efficient convolutions, data structures like octrees [29], voxels [22] or high-dimensional lattices [33] are utilized. These works use sophisticated strategies to avoid redundant computations. However, the required data preprocessing can be time consuming and computationally expensive, especially for larger point clouds.

To represent and process large scale point clouds Landrieu and Simonovsky [16] introduce superpoint graphs (SPGs). They transfer the idea of superpixels [2] to point clouds and propose a geometric pre-partitioning of the data into simple primitives. The resulting *superpoints* are modeled together with derived features within the SPG and processed with [32].

2.2. Pseudo-3D

All considered approaches so far are designed to process sceneries, where objects are fully described in 3D space (*i.e.*

both the front and back of an object are reconstructed by the point cloud). However, a single LiDAR sweep just measures depth originating from the sensor center. Thus, it generates a 2.5D representation, where only the surface parts of an object facing the LiDAR are visible. While the point cloud is sparse, in 3D and when projected onto the RGB image plane, a dense representation can be obtained by considering the native properties of the sensor (see Section 3.1 for details).

As common LiDARs have a nearly constant horizontal angle resolution, dense representations can be obtained via cylindrical projection [18, 5, 24] or spherical projection [37, 35, 38]. However, in practice the vertical resolution is not constant. For example, the Velodyne HDL-64E laser scanner (used by the KITTI benchmark) sweeps 64 beams with approximately two different angular distances. The top set of 32 beams has a higher angular distance between subsequent beams than the bottom set. Other LiDARs (*e.g.* Velodyne VLP-32C) sample denser near the horizon to improve long-range detections.

Our work is based on SqueezeSeg [37] by Wu *et al.*, an adaptation of SqueezeNet [14] for LiDAR point cloud segmentation. It uses a spherical projection to obtain a dense representation of the LiDAR point cloud and encodes 3D coordinates, range and reflectance intensity into the channels of the input image. In [37], they synthesize large amounts of point cloud data utilizing Grand Theft Auto V (GTA-V), a famous video game, to increase its performance on KITTI's *car* class. This synthetic data, however, does not sufficiently represent the other classes realistically, because the underlying geometry has been excessively simplified within the game. For example, the torso, head and limbs of pedestrians within GTA-V are crudely modeled as cylinders. In our work we do not rely on massively generated synthetic data and still achieve state-of-the-art results in real time.

2.3. RGB/3D Fusion

When depth information is densely available and properly registered with RGB imagery, it is an obvious choice to improve results on different vision tasks. Gupta *et al.* [11] propose three handcrafted auxiliary channels derived from depth to improve segmentation compared to a single depth channel. Hazirbas *et al.* [13] use a separate network branch for depth to improve results compared to an equivalent single branch architectures with additional input channels. Recently, Zeng *et al.* [40] use two network branches to estimate surface normals. Similar to these approaches we fuse the respective features at multiple layers as well. However, since depth is not densely available given a LiDAR point cloud, element-wise operations like summation are not sufficient. We introduce a progressive fusion scheme, based

on polyharmonic spline interpolation [8] to overcome this issue efficiently.

Recently, various works utilize both RGB and LiDAR data, mostly for the task of 3D object detection. For example, the Multi-View 3D network (MV3D) [5] by Chen *et al.* maps the LiDAR point cloud to a bird's eye view (BEV) to generate object proposals. Given these proposals, features from the BEV, a cylindrical LiDAR projection and an RGB image branch are fused to classify an object and regress its bounding box. In Frustum PointNets [25], Qi *et al.* use Faster R-CNN [28] to create 2D proposals from RGB imagery. The result is propagated to 3D space and refined. Except for the object class, there is no further information exchange between the RGB and the 3D detection head. Both works rely on proposals from a single data modality and thus, are prone to loose objects, because they are not using all available information from the beginning on. Ku *et al.* [15] propose *Aggregate View Object Detection (AVOD)*, a network based on RGB and BEV features. However, they evaluate a predefined set of 3D anchor boxes and thus, are limited by their predefined choice.

Liang *et al.* [21] propose a feature warping from an RGB CNN branch to a LiDAR BEV. To this end, they need to perform a k-nearest neighbor search in the point cloud for each pixel in the BEV image. However, with the distance to the sensor the point cloud becomes increasingly sparse. In [20] they mitigate this issue utilizing an auxiliary depth completion task.

However, in contrast to these works, we use two native and dense representations which can be processed by standard CNNs without any further preprocessing. Thereby, we are able to densely warp and fuse the features and leverage all information jointly as early as possible.

3. FuseSeg

In this section we describe the proposed feature warping module and how we extend SqueezeSeg in order to utilize RGB information. In particular, rather than warping the RGB image into the range space, we apply an ImageNet CNN directly on the undistorted input images. Consequently, we can leverage the benefits of transfer learning better, as objects are not distorted in the original RGB input. We then fuse RGB features extracted at multiple layers of the ImageNet CNN (MobileNetV2) into the segmentation architecture.

In order to align the RGB features with the range features for segmentation, we warp them by leveraging the correspondences available due to the calibrated setup. Subsequently, the warped RGB features are concatenated with features from the range image to perform segmentation.

Figure 1 schematically illustrates our network architecture and the feature warping. For efficiency, we subsample point correspondences (*control points*) within the different

input images. In the following, we discuss the discretization of the LiDAR point cloud (Section 3.1), the foundation of our architecture *SqueezeSeg* (Section 3.2) and the warping procedure (Section 3.3) in more detail.

3.1. LiDAR Geometry

A common LiDAR sensor dedicated for autonomous driving purposes sends out multiple vertically distributed beams and determines the distance to the first hit object by measuring the time-of-flight until the reflection is detected. A $360°$ recording is usually obtained by a steady rotation of the laser transmitter itself or a respective deviation *e.g.*, via mirrors.

SqueezeSeg processes the resulting point cloud on a spherical grid by discretizing the azimuth ϕ and zenith θ of each 3D point (x, y, z) by

$$\phi = \arcsin \frac{y}{\sqrt{x^2 + y^2}}, \quad \tilde{\phi} = \lfloor \phi/\Delta\phi \rfloor, \qquad (1)$$

$$\theta = \arcsin \frac{z}{\sqrt{x^2 + y^2 + z^2}}, \quad \tilde{\theta} = \lfloor \theta/\Delta\theta \rfloor, \qquad (2)$$

where $\Delta\phi$ and $\Delta\theta$ denote the discretization resolution and $(\tilde{\phi}, \tilde{\theta})$ the coordinates on the spherical grid, respectively. The resulting spherical image constitutes a dense representation, which can be processed by a CNN. It incorporates five channels, the Cartesian point coordinates (x, y, z), range $r = \sqrt{x^2 + y^2 + z^2}$ and the LiDAR's reflectance intensity measurement. Unless stated otherwise, we adopt this channel configuration.

However, in practice the vertical resolution $\Delta\theta$, which is the angle between subsequent LiDAR beams is not constant. Thus, we adapt the representation from [23] and utilize the *beam id* to assign each point to its row $\tilde{\theta}$ in the image. The *beam id* can be easily retrieved from the LiDAR sensor. This allows for an unambiguous vertical discretization to obtain a dense native *range representation*, which we use as the *laser range image*. This range representation is even easier to obtain than the spherical one (*i.e.* no need for zenith projection) and reduces holes and coordinate conflicts in the data. If (due to the horizontal discretization) multiple 3D points fall onto to the same pixel in the range image, we choose the one with azimuth position nearest to the respective pixel center.

3.2. SqueezeSeg

We base our architecture on SqueezeSeg [37]. It is a lightweight architecture based on SqueezeNet [14], specifically designed to segment spherical images. It adapts the *FireModule* layers from [14] and introduces related *FireDeconvs* instead of using convolutions and transposed-convolutions in order to reduce computational effort.

Similar to [3], SqueezeSeg uses a conditional random field (CRF) in order to refine the segmentation results es-

pecially at the object borders. The CRF penalizes assigning different labels to similar points in terms of angular and Cartesian coordinates. In other words, points with nearby coordinates in the range image as well as in 3D space are dedicated to get the same label.

Finally, it minimizes a pixel-wise cross-entropy loss. To mitigate the impact of the class imbalance, cyclists and pedestrians are stronger weighted. Furthermore, outliers, due to failed laser measurements, are masked out during loss computation.

3.3. Multi-modal Feature Fusion

In order to merge RGB features from a CNN layer with those from the laser range image, we propose to use the known calibration of LiDAR and RGB camera. We illustrate this process in Figure 2. For each valid pixel in the range image, the corresponding 3D position of the laser point is available. Given the 3×4 projection matrix $\mathbf{P}$, we can project the 3D coordinates onto the image via

$$\mathbf{x} = \mathbf{PX}, \qquad (3)$$

where $\mathbf{X}$ and $\mathbf{x}$ denote homogeneous 3D and pixel coordinates [12], respectively. The projection matrix itself can be easily derived from the RGB camera calibration and the transformation form LiDAR to camera coordinate system.

Points visible in both, the RGB and range image denote correspondences between the two representations. A naïve approach would be to use these correspondences to look up every 3D point's color within the RGB image and thereby colorize the range image.

However, the comparably dense and valuable information provided by the RGB image would be left unused. Thus, we propose to fuse the intermediate feature representations extracted from respective CNNs. We use well studied architectures [30, 37] capable of providing useful feature representations for both, the RGB and range image. We extract and warp RGB features at multiple levels of the network such that they align with their range counterparts. We map the ImageNet features from the 7th, 14th and 19th layer of MobileNetV2 to the layers *Fire2*, *Fire4* and *Fire7* of SqueezeSeg, respectively. We choose the layers before a pooling operation in MobileNetV2 and warp into similar sized SqueezeSeg layers whilst avoiding the ones which are passed through the skip connections. As a consequence, we exploit the RGB features with the highest representational capabilities of the respective spatial resolution and save parameters within the decoder. Using different or less connection points leads to slightly inferior results.

Since we warp feature tensors at different network layers (instead of raw input images), we cannot rely on a simple lookup. This is due to the fact that we do not have explicit correspondences between positions within the range feature
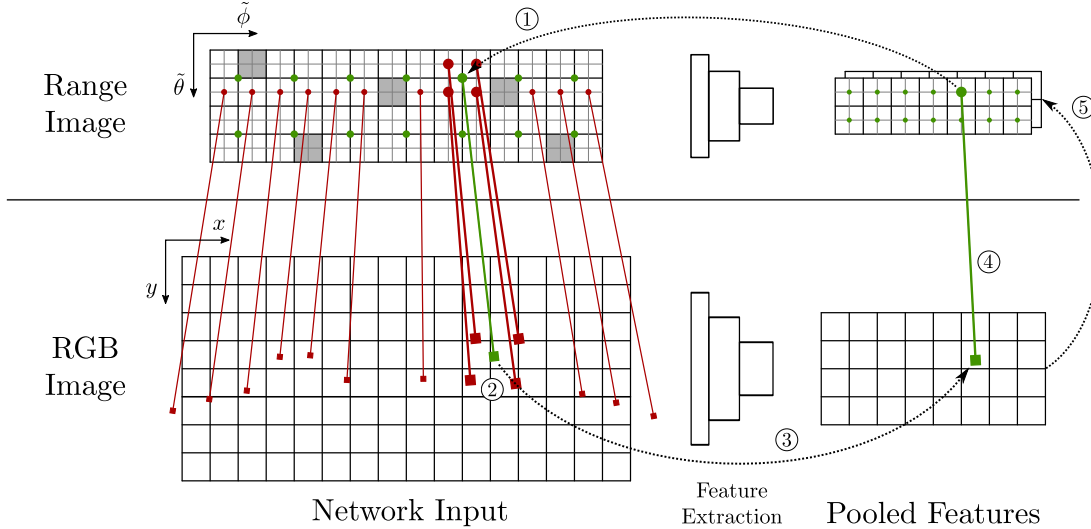
Figure 2: Illustration of the warping process at a specific feature extraction layer (right). To align the RGB features (bottom) with the range features (top), we first ① compute the range image location corresponding to the current range feature (green dots). Given the point correspondences (red) between the range and RGB image, ② we use a first-order polyharmonic spline interpolation for sub-pixel sampling of the correct RGB position (green cube). Then, ③ we compute the respective position within the RGB feature space to obtain the feature correspondence ④. Given that, ⑤ we are able to densely warp the RGB features such that they spatially align with the range features. Concatenating them allows for jointly leveraging both information cues for arbitrary 3D perception tasks. The gray pixels denote laser outliers (*e.g.* due to transparent surfaces).

tensor and their counterparts within the RGB feature tensor. For proper feature warping, we need sub-pixel accuracy (see green line segments in Figure 2). Additionally, we need to deal with laser measurement outliers (e.g. due to transparent surfaces or far distant objects) which cause missing range image-to-RGB correspondences.

To address these issues, we treat the range image-to-RGB correspondences and their positions as *control points* for a first-order polyharmonic spline interpolation [8]. Passing query positions $\mathbf{x}$ in the range image, we obtain the corresponding interpolated position $f(\mathbf{x})$ in the RGB image with

$$f(\mathbf{x}) = \sum_{i=1}^{N} \mathbf{w}_i \, \|\mathbf{x} - \mathbf{c}_i\|_2 + \mathbf{V}^{\mathrm{T}} \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}, \qquad (4)$$

where $\mathbf{c}_i$ are the N range pixel coordinates with valid corresponding positions $f(\mathbf{c}_i)$ in the RGB image. By solving a linear system of equations, we obtain the interpolating spline weights $\mathbf{w}_i$ and $\mathbf{V}$. Note that we need to do this computation only once for each sample and we can reuse the weights for all interleaved layers.

In order to retrieve correspondences for a specific spatial resolution, we scale the pixel positions within the range features such that they are aligned with the original input image. Subsequently, we sample the corresponding position in the RGB space using the calculated spline interpolation. This yields the sub-pixel accurate position within the input

RGB image for each pixel in the range feature tensor. From this, we can retrieve the corresponding position within the RGB feature tensor as shown in Figure 2.

To derive the actual value at the non-discrete position in RGB feature space, we bilinearly interpolate the four nearest neighboring features. The part of the warped feature tensor with correspondences outside the RGB image is set to zero.

4. Experiments

We evaluate our method on KITTI [10, 9] and reuse the train/val-split from [37]. We also follow their training protocol and adopt their parameters: We consider the three main classes *cars*, *pedestrians* and *cyclists* and add an auxiliary class to model the background. KITTI provides labels in the horizontal field of view of $90°$ only, thus we limit our consideration to this area. Additionally, our range images do have the same resolution of 512×64 and, unless otherwise stated, the same input channels as in [37].

We augment the data by random horizontal flips and slight deviations in saturation, contrast and brightness of the RGB image. Based on a checkpoint trained with Li-DAR features, we re-initialize the respective weights and fine-tune the network. We implement our framework in TensorFlow [1] and use a GeForce GTX 1080Ti GPU for all runtime evaluations.

In the following, we evaluate the effect of our proposed
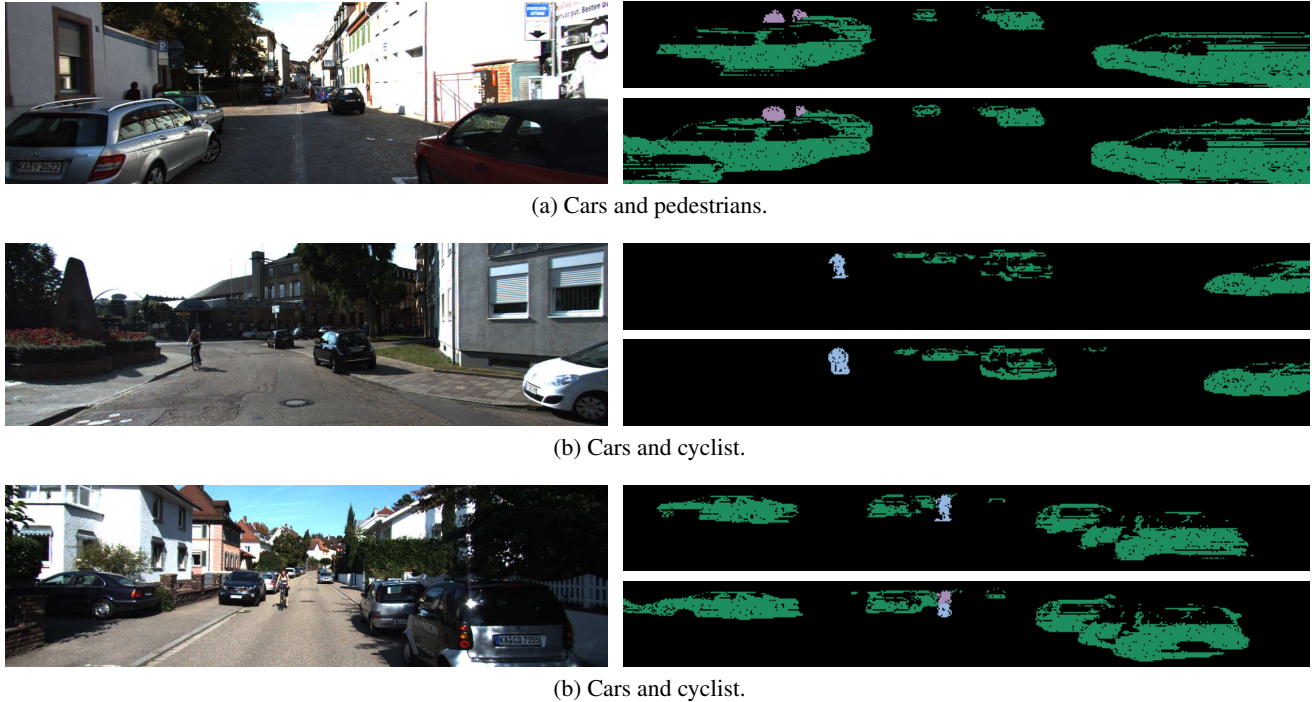
(a) Cars and pedestrians.

(b) Cars and cyclist.

(b) Cars and cyclist.

Figure 3: Qualitative results of FuseSeg. We show the RGB input (left), the ground truth (top right) and the prediction of the network (bottom right). We detect even small and partially occluded objects (a,b) as well as objects outside the RGB image and unlabeled in the lower corners of the range image (a). Sometimes a cyclist is detected separately from the bicycle (c).

Method	car	ped	cyc	avg	rt [ms]
FuseSeg	<u>71.1</u>	**36.8**	**36.0**	**48.0**	20
FuseSeg R-RGB	67.4	23.4	31.2	40.7	20
SqSeg w/o RGB †	67.2	20.2	24.1	37.2	9
SqSeg w/ RGB	63.7	18.8	22.8	35.1	13
PointSeg [35] *	67.4	19.2	32.7	39.8	-
SqSeg [37] *	64.6	21.8	25.1	37.2	13.5
SqSegV2 [38] *	**73.2**	<u>27.8</u>	<u>33.6</u>	<u>44.9</u>	-

Table 1: Point cloud segmentation performance (IoU in %) and runtime (in milliseconds) on KITTI. To show the effectiveness of our feature fusion we compare with vanilla SqueezeSeg with color as additional input channels. Results marked * are taken from the respective paper and † mark our reproduced results. Scores and runtime for *SqSeg w/o RGB* differ slightly from [37] as we retrain it for a fair comparison on our GPU.

FuseSeg method on point cloud segmentation in comparison with state-of-the-art methods (Section 4.1). Subsequently, we compare the architecture with RGB semantic segmentation networks to validate our warping-based feature fusion (Section 4.2). Finally, we show that we can reduce the number of control points and the accompanied computational cost without negatively affecting the performance (Section 4.3).

4.1. Feature Fusion

We show the merit of the fused image features by comparing it not only with SqueezeSeg, but also with state-of-the-art point cloud segmentation methods. Table 1 shows the results for all three relevant object classes and the respective runtime, while Figure 3 shows some qualitative results. We report the best average intersection-over-union over all three classes.

To provide an additional baseline, we also pass the RGB channels to SqueezeSeg (*SqSeg w/ RGB*). Thus, we colorize its range representation. To this end, we project each point onto the RGB image and sample the underlying pixel's color. Note that not the entire range image is colored, only those 3D points which are visible in the RGB image.

The additional color channels even lower the performance of SqueezeSeg. The reason for this drop is that SqueezeSeg is optimized for runtime speed. Consequently its representational power is not able to process all information. Since we utilize a separate lightweight network to process the RGB information, we introduce another baseline (*FuseSeg R-RGB*): We warp the RGB image to its range counterpart (see Figure 5 for an upscaled example) and pass it to our RGB branch. Note that this baseline has the same number of parameters as FuseSeg.

As we see in our experiments, using a pre-trained ImageNet CNN/MobileNetV2 for extracting features in a
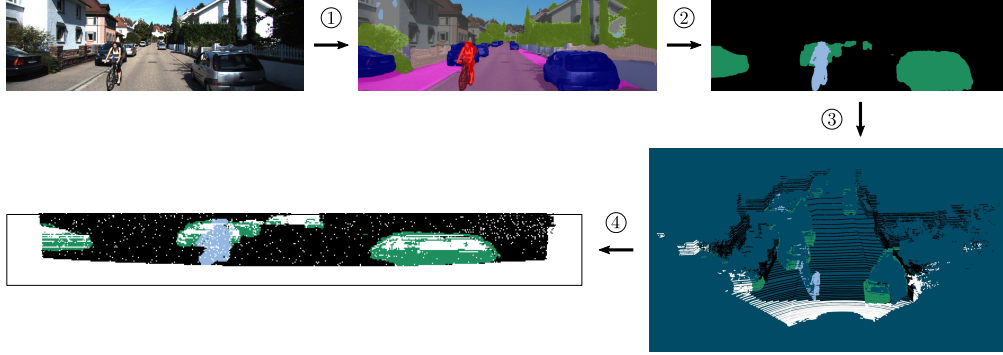
Figure 4: Illustration of the evaluation process solely based on RGB. We infer a semantic mask from the RGB image ① using a segmentation network trained on KITTI and CityScapes [7]. Subsequently, we fused neighboring *rider* and *bicycle* regions to *cyclist* in order to obtain a compatible annotation policy ②. Given the calibrated setup and thus, the projection matrix (see Eq. 3) we are able to lookup a class ③ for each point visible in the RGB image (black denotes background, white denotes points with projections outside the RGB image and thus, no derived class). We evaluate the resulting range image ④ (only on the non-white area) and compare it with our method. We show that leveraging depth information using our fusion method significantly outperforms RGB based methods with comparable feature extraction backends, whilst being almost five times faster (see Table 2).

warped range image already benefits segmentation performance compared to using no ImageNet CNN for the RGB information. Further, by using our proposed warping method to fuse on the feature level instead of the (RGB) input level, we further significantly improve accuracy. The main reason for this is that the warped RGB input representation is heavily distorted and thus impairs the performance of ImageNet features. In contrast, with our approach the ImageNet CNN operates on an undistorted RGB input on which it better benefits from transfer learning.

FuseSeg improves segmentation, especially on the smaller classes *pedestrian* and *cyclist*, by a large margin. We increase the mean intersection-over-union (IoU) by 18% respectively 13.2% compared to SqueezeSeg. We even outperform its successor SqueezeSegV2 [38] on average by 3.1%, which could be improved by our approach as well.

4.2. FuseSeg vs RGB Semantic Segmentation Approaches

In order to show the effectiveness of our warping-based feature fusion, we compare our approach with semantic segmentation approaches solely relying on RGB information. More specifically, we compare FuseSeg with DeepLabv3+ [4] in combination with two feature extraction backends, a MobileNetV2 [30] and a more powerful Xception65 [6] feature extractor. We infer that outperforming equivalent state-of-the-art architectures validated our fusion approach. Figure 4 illustrates the process of deriving and evaluating labeled point clouds from RGB segmentation masks.

We fine-tune the pre-trained DeepLabv3+ models on CityScapes and the KITTI semantic segmentation data and

Method	car	ped	cyc	avg	rt [ms]
DLv3+ MNV2	66.9	33.8	30.2	43.6	95
DLv3+ Xception65	71.3	**41.4**	37.4	50.0	369
FuseSeg	**73.7**	39.7	**41.2**	**52.1**	**20**

Table 2: Segmentation performance (IoU in %) and runtime (in milliseconds) on KITTI. FuseSeg compared with RGB-based semantic segmentation network (DeepLabv3+) trained on both, CityScapes [7] and the KITTI segmentation benchmark. Given the registration, LiDAR points are projected onto the image and classified according to their position in the segmentation mask. We outperform the respective MobileNetV2 (MNV2) DeepLabv3+ by a large margin for all classes and even the much more powerful Xception65 backend on average. Thereby, our architecture is almost five times as fast as the DeepLabv3+ MNV2 counterpart and eighteen times as fast as the Xception65 pendant.

ensure that no image of our validation set is used for training. We trained until convergence and choose the checkpoint with the best segmentation result on the KITTI validation set. To overcome the diverging annotation policies of the two datasets, we fuse neighboring *bicycle* and *rider* regions to *cyclist*.

We create segmentation masks for each RGB image by passing it through the trained models and segment the 3D points by projecting them onto the masks (see Eq. 3). All classes except *car*, *bicycle* and *pedestrian* are considered as background. Thus, we segment the point clouds without using any depth information. For this comparison, we only evaluate the part of the range image with color information for all methods (Thus, the evaluation region differs from Section 4.1).

Figure 5: Illustration of warping artifacts due to the baseline between RGB camera and LiDAR sensor. In order to visualize possible artifacts (here *e.g.* cyclist and van roof) we warp the RGB image to its range pendant (we upscale the control points of the range image by a factor of two for visibility). The number and thus, position of control points influence these distortions.

Table 2 shows the IoU on the respective classes and the runtime of each method. While we clearly outperform DeepLabv3+ in terms of runtime, we outperform the network based on MobileNetV2 on all three classes. Note, that this is the same backend as used in FuseSeg for RGB information. As a consequence, this demonstrates that depth adds valuable information to the segmentation task and our fusion approach is an effective and very efficient method to utilize it.

We are even better than the powerful Xception65 DeepLabv3+ on average performance, despite using the weaker backend. Our modular design allows the exchange of the RGB backend in a plug-and-play manner, but one of our research goals is real-time speed leading to the choice of MobileNetV2.

4.3. Number of Control Points

# Ctrl Pts	car	ped	cyc	avg	rt [ms]
4	69.9	33.0	33.9	45.6	19
24	70.4	36.2	**36.7**	47.7	19
48	**71.1**	**36.8**	36.0	**48.0**	20
96	70.7	36.0	33.8	46.8	20
192	71.0	35.3	35.2	47.2	22
384	70.7	36.6	36.0	47.7	26

Table 3: Segmentation Performance (IoU in %) and runtime (in milliseconds) of FuseSeg on KITTI. We compared different amounts of control points and report best average IoU performance and runtime. While the computational effort of an inference step linearly increases with the amount of control points, performance saturates.

In KITTI there are up to 19k point correspondences between an RGB image and range representation. However, since computational cost of the interpolation increases with the number of control points, a small number of control points is desirable. To this end, to obtain a good coverage in the target domain, we perform FPS on the coordinates in the range image (in contrast to FPS on 3D coordinates) to reduce the number of control points.

We compare different configurations aiming at a reliable assessment. We vary the number of control points used by our architecture and evaluate segmentation accuracy as well as runtime. Table 3 shows the speed-vs-accuracy trade-off. Interestingly, we only need a very small number of control points, *i.e.* 24, to estimate a decent warping and achieve state-of-the-art results. We see that there is no notable variation of the accuracy for the *car* class, which can be explained by their size.

However, for smaller objects, *i.e. pedestrians* and *cyclist*, we observe a notable sensitivity regarding the control points and multiple spikes at certain point numbers. Due to the baseline between camera and LiDAR and the resulting parallax, a flawless warping is not always possible. This distortion peaks at high depth differences, *e.g.* at the edges of visible objects (see Figure 5). We hypothesize that a certain number of control points favors these distortions more than others. More elaborate sampling methods, *e.g.* focusing on depth discontinuities within the range image might mitigate these sensitivities, but are out of the scope of this paper.

5. Conclusion

We propose a simple and effective way to leverage RGB features for LiDAR point cloud segmentation. Utilizing the *range representation* of LiDAR point clouds allows us to process them with known CNN strategies. Then, our efficient warping-based feature fusion enables us to use the benefits of transfer learning on the dense and rich information provided by RGB data jointly with features derived from LiDAR data. Thereby, we still fulfill real-time requirements, performing at 50 fps. This is twice as fast as the recording speed of today's LiDAR sensors. Thus, our method can easily be utilized in autonomous cars and robots.

Furthermore, the encoder of FuseSeg is applicable as feature extractor for various 3D perception tasks. Finally, our warping strategy in combination with the range representation can be used to interleave features in both directions and thus, also improve RGB-based object detection and semantic segmentation.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. In *Proc. OSDI*, 2016.

[2] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *TPAMI*, 34(11):2274–2282, 2012.

[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *TPAMI*, 40(4):834–848, 2017.

[4] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In *Proc. ECCV*, 2018.

[5] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-View 3D Object Detection Network for Autonomous Driving. In *Proc. CVPR*, 2017.

[6] F. Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In *Proc. CVPR*, 2017.

[7] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. CVPR*, 2016.

[8] G. E. Fasshauer. *Meshfree Approximation Methods with MATLAB*, volume 6. World Scientific, 2007.

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets Robotics: The KITTI Dataset. *IJRR*, 32(11):1231–1237, 2013.

[10] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. CVPR*, 2012.

[11] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning Rich Features from RGB-D Images for Object Detection and Segmentation. In *Proc. ECCV*, 2014.

[12] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[13] C. Hazirbas, L. Ma, C. Domokos, and D. Cremers. FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-Based CNN Architecture. In *Proc. ACCV*, 2016.

[14] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

[15] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander. Joint 3D Proposal Generation and Object Detection from View Aggregation. In *Proc. IROS*, 2018.

[16] L. Landrieu and M. Simonovsky. Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs. In *Proc. CVPR*, 2018.

[17] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. PointPillars: Fast Encoders for Object Detection from Point Clouds. In *Proc. CVPR*, 2019.

[18] B. Li, T. Zhang, and T. Xia. Vehicle Detection from 3D Lidar Using Fully Convolutional Network. In *Proc. RSS*, 2016.

[19] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. PointCNN: Convolution On X-Transformed Points. In *Proc. NeurIPS*, 2018.

[20] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun. Multi-Task Multi-Sensor Fusion for 3D Object Detection. In *Proc. CVPR*, 2019.

[21] M. Liang, B. Yang, S. Wang, and R. Urtasun. Deep Continuous Fusion for Multi-Sensor 3D Object Detection. In *Proc. ECCV*, 2018.

[22] D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *Proc. IROS*, 2015.

[23] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington. LaserNet: An Efficient Probabilistic 3D Object Detector for Autonomous Driving. *arXiv preprint arXiv:1903.08701*, 2019.

[24] K. Minemura, H. Liau, A. Monrroy, and S. Kato. LMNet: Real-time Multiclass Object Detection on CPU Using 3D LiDAR. In *Proc. ACIRS*, 2018.

[25] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. In *Proc. CVPR*, 2018.

[26] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proc. CVPR*, 2017.

[27] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. NeurIPS*, 2017.

[28] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Proc. NeurIPS*, 2015.

[29] G. Riegler, A. Osman Ulusoy, and A. Geiger. OctNet: Learning Deep 3D Representations at High Resolutions. In *Proc. CVPR*, 2017.

[30] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proc. CVPR*, 2018.

[31] S. Shi, X. Wang, and H. Li. PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud. *arXiv preprint arXiv:1812.04244*, 2018.

[32] M. Simonovsky and N. Komodakis. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *Proc. CVPR*, 2017.

[33] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz. SPLATNet: Sparse Lattice Networks for Point Cloud Processing. In *Proc. CVPR*, 2018.

[34] S. Wang, S. Suo, W.-C. Ma, A. Pokrovsky, and R. Urtasun. Deep Parametric Continuous Convolutional Neural Networks. In *Proc. CVPR*, 2018.

[35] Y. Wang, T. Shi, P. Yun, L. Tai, and M. Liu. PointSeg: Real-Time Semantic Segmentation Based on 3D LiDAR Point Cloud. *arXiv preprint arXiv:1807.06288*, 2018.

[36] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *TOG*, 2019.

[37] B. Wu, A. Wan, X. Yue, and K. Keutzer. SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time

Road-Object Segmentation from 3D LiDAR Point Cloud. In *Proc. ICRA*, 2018.

[38] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer. Squeeze-SegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud. In *Proc. ICRA*, 2019.

[39] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao. SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters. In *Proc. ECCV*, 2018.

[40] J. Zeng, Y. Tong, Y. Huang, Q. Yan, W. Sun, J. Chen, and Y. Wang. Deep Surface Normal Estimation with Hierarchical RGB-D Fusion. In *Proc. CVPR*, 2019.