RangeNet++: Fast and Accurate LiDAR Semantic Segmentation

Andres Milioto Ignacio Vizzo Jens Behley Cyrill Stachniss

***Abstract*— Perception in autonomous vehicles is often carried out through a suite of different sensing modalities. Given the massive amount of openly available labeled RGB data and the advent of high-quality deep learning algorithms for image-based recognition, high-level semantic perception tasks are pre-dominantly solved using high-resolution cameras. As a result of that, other sensor modalities potentially useful for this task are often ignored. In this paper, we push the state of the art in LiDAR-only semantic segmentation forward in order to provide another independent source of semantic information to the vehicle. Our approach can accurately perform full semantic segmentation of LiDAR point clouds at sensor frame rate. We exploit range images as an intermediate representation in combination with a Convolutional Neural Network (CNN) exploiting the rotating LiDAR sensor model. To obtain accurate results, we propose a novel post-processing algorithm that deals with problems arising from this intermediate representation such as discretization errors and blurry CNN outputs. We implemented and thoroughly evaluated our approach including several comparisons to the state of the art. Our experiments show that our approach outperforms state-of-the-art approaches, while still running online on a single embedded GPU. The code can be accessed at `https://github.com/PRBonn/lidar-bonnetal`.**

I. Introduction

Semantic scene understanding is one of the key building blocks of autonomous robots working in dynamic, real-world environments. To achieve the required scene understanding, robots are often equipped with multiple sensors that allow them to leverage the strengths of each modality. Combining multiple complementary sensing modalities allows for covering the shortcomings of individual sensors such as cameras, laser scanners, or radars. This is particularly critical in the context of autonomous driving, where a failure of one modality can have lethal or significant monetary consequences in case it is not properly covered by another redundant sensor.

An important tasks in semantic scene understanding is the task of semantic segmentation. Semantic segmentation assigns a class label to each data point in the input modality, i.e., to a pixel in case of a camera or to a 3D point obtained by a LiDAR. In this paper, we explicitly address semantic segmentation for rotating 3D LiDARs such as the commonly used Velodyne scanners. Unfortunately, the majority of state-of-the-art methods currently available for semantic segmentation on LiDAR data either don't have enough representational capacity to tackle the task, or are computationally too expensive to operate at frame-rate on a mobile GPU. This makes them not suitable to aid the task of supporting autonomous vehicles, and addressing these issues is the aim of this work.

All authors are with the University of Bonn, Germany. This work has partly been supported by the German Research Foundation under Germany's Excellence Strategy, EXC-2070 - 390732324 (PhenoRob) as well as grant number BE 5996/1-1, and by NVIDIA Corporation.

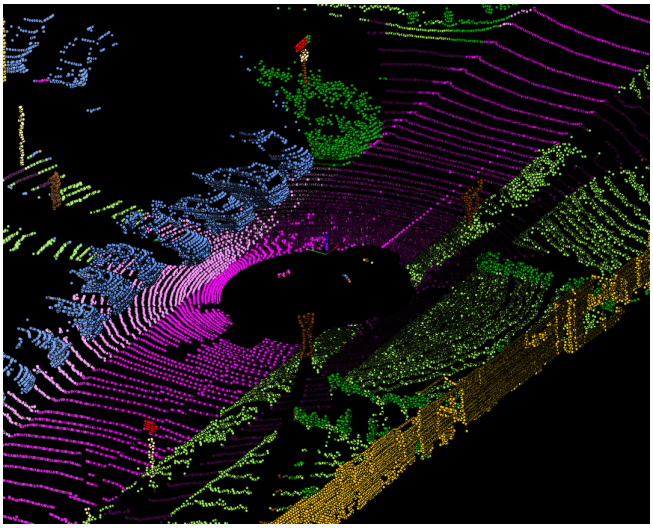

Fig. 1: Velodyne HDL-64E laser scan from KITTI dataset [7] with semantic information from RangeNet++. Best viewed in color, each color represents a different semantic class.

The main contribution of this paper is a new method for accurate, fast, LiDAR-only semantic segmentation. We achieve this by operating on a spherical projection of the input point cloud, i.e., a 2D image representation, similar to a range image, and therefore exploit the way the points are detected by a rotating LiDAR sensor. Our method infers the full semantic segmentation for each pixel of the image using any CNN as a backbone. This yields an efficient approach but can lead to issues caused by discretization or blurry CNN outputs. We effectively resolve these issues via a reconstruction of the original point with semantics without discarding any points from the original point cloud, regardless of the used resolution of the image-based CNN. This post-processing step, which also runs online, operates on the image representation and is tailored towards efficiency. We can calculate nearest neighbors in constant time for each point and exploit GPU-based calculations. This allows us to infer full semantic segmentation of LiDAR point clouds accurately and faster than the frame rate of the sensor. Since the approach runs with any range image-based CNN backbone, we call it RangeNet++. See Fig. 1 for an example.

In sum, we make three key claims: Our approach is able to (i) accurately perform semantic segmentation of LiDAR-only point clouds, surpassing the state of the art significantly, (ii) infer semantic labels for the complete original point

cloud, avoiding to discard points regardless of the level of discretization used in the CNN, and (iii) work at the frame rate of a Velodyne scanner on an embedded computer that can easily fit in robots or in a vehicle.

II. Related Work

Semantic segmentation for autonomous driving using images made an immense progress in recent years due to the advent of deep learning and the availability of increasingly large-scale datasets for the task, such as CamVid [2], Cityscapes [4], or Mapillary [12]. Together, this enables the generation of complex deep neural network architectures with millions of parameters achieving high-quality results. Prominent examples are Deeplab V3 [3] and PSPNet [23].

Despite their impressive results, these architectures are too computationally expensive to run in real-time on an autonomous system, which is a must for autonomous navigation exploiting semantic cues. This spawned the creation of more efficient approaches such as Bonnet [11], ENet [13], ERFNet [17], and Mobilenets V2 [18], which leverage the law of diminishing returns to find the best trade-off between runtime, the number of parameters, and accuracy. These, however, are designed for images and not for LiDAR scans.

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

To tackle the problem of the lack of data, Wu *et al.* [21], [22] used the provided bounding box of the KITTI dataset [7]. They also leveraged simulation to generate realistic looking scans from a game engine. We have released the first large-scale dataset with full semantic segmentation of LiDAR scans [1], in which all scans of the KITTI odometry dataset [7] were densely annotated, i.e., over $43\,000$ scans, with over 3.5 billion annotated points. Without the data-starvation barrier, this paper investigates which of the current state-of-the-art algorithms can be exploited and adapted for point cloud in the autonomous driving context.

Leveraging large datasets for other contexts [5], [8], several deep learning-based methods for 3D semantic segmentation were recently developed, such as PointNet [14], PointNet++ [15], TangentConvolutions [20], SPLATNet [19], SuperPointGraph [10], and SqueezeSeg [21], [22].

One of the problems of dealing with point cloud data directly is the lack of a proper ordering, which makes learning order-invariant feature extractors extremely challenging. Qi *et al.* [14], [15] use as inputs the raw, un-orderered point clouds and apply symmetrical operators that are able to deal with this ordering problem. For this purpose, max pooling is used by PointNet [14] to combine the features and generate permutation-invariant feature extractors. This, however, is a limiting factor for PointNet, causing it to lose the ability to capture spatial relationships between features. This limits its applicability to complex scenes. PointNet++ [15] tackles this problem by using a hierarchical approach for feature extraction. By exploiting individual PointNets in a local vicinity, it captures short-range dependencies and then applies this concept hierarchically to capture global dependencies.

Tatarchencko *et al.* [20] take a different approach to handle unstructured point clouds. They propose TangentConvolutions that apply CNNs directly on surfaces, which can only be achieved if neighboring points are sampled from the same surface. In this case, the authors can define a tangent convolution as a planar convolution that is applied to the projection of the surface at each point. This assumption is, however, violated in case of a rotating LiDAR and the generated distance-dependent sparsity of the point cloud.

Su *et al.* [19] approach the representational problem differently in SPLATNet, by projecting the points in a high-dimensional sparse lattice. However, this approach does not scale well neither in terms of computation nor in memory consumption. To alleviate this, bilateral convolutions [9] allow them to apply theses operators exclusively on occupied sectors of the lattice.

Landrieu *et al.* [10] manage to summarize the local relationships in a similar fashion to PointNets by defining a SuperPoint Graph. This is achieved by creating so-called SuperPoints, which are locally coherent, geometrically homogeneous groups of points that get embedded by a PointNet. They create a graph of SuperPoints that is an augmented version of the original point cloud, and train a graph convolutional network to encode the global relationships.

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

Leading the charge in online processing, SqueezeSeg and SqueezeSegV2 [21], [22], by Wu *et al.*, also use a spherical projection of the point cloud enabling the usage of 2D convolutions. Furthermore, a light-weight fully convolutional semantic segmentation is applied along with a conditional random field (CRF) to smooth the results. The last step is an un-discretization of the points from the range image back into the 3D world. Both are capable of running faster than the sensor rate, i.e., $10\,\text{Hz}$, and we use them as the basis of our approach.

Several limitations need to be addressed in order to provide full semantic segmentation with this framework. First, the projection needs to be extended to include the full LiDAR scan, since the SqueezeSeg framework only uses the frontal 90 degrees of the scan, where the objects of the original KITTI dataset labels are annotated by bounding boxes. Second, the SqueezeNet backbone is not descriptive enough to infer all the 19 semantic classes provided by our dataset [1]. Third, we replace the CRF, which operates in the image domain by an efficient, GPU-based nearest neighbor search acting directly on the full, un-ordered point cloud. This last step enables the retrieval of labels for all points in the cloud, even if they are not directly represented in the range image, regardless of the used resolution.

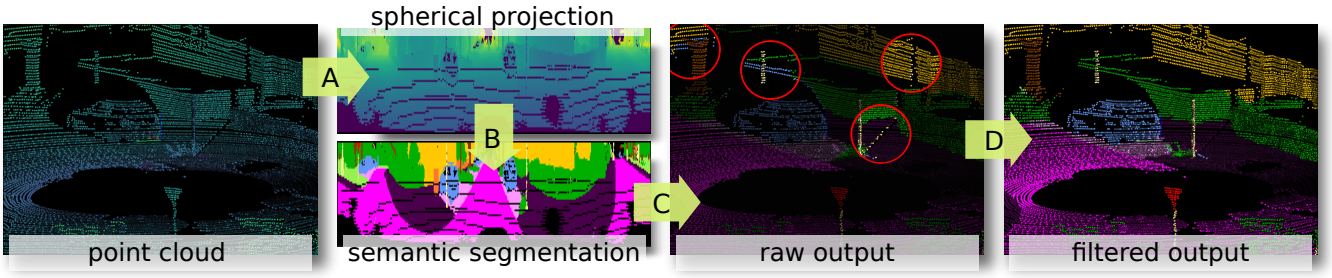

Fig. 2: Block diagram of the approach. Each of the arrows corresponds to one of our modules.

We propose a novel approach inspired by projection-based methods which allow the usage of planar convolutions and overcomes its drawbacks. Our method accurately segments entire LiDAR scans at or faster than the frame rate of the sensor (around 10 Hz), uses range images and 2D CNNs as a proxy, and deals properly with the discretization errors that need to be addressed after re-projecting the results to the 3D point cloud.

III. Our Approach

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

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

A. Range Image Point Cloud Proxy Representation

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

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

where (u, v) are said image coordinates, (h, w) are the height and width of the desired range image representation, $\mathrm{f} = \mathrm{f}_{\mathrm{up}} + \mathrm{f}_{\mathrm{down}}$ is the vertical field-of-view of the sensor, and $r = ||\mathbf{p}_i||_2$ is the range of each point. This procedure results in a list of (u, v) tuples containing a pair of image coordinates for each $\mathbf{p}_i$, which we use to generate our proxy representation. Using these indexes, we extract for each $\mathbf{p}_i$, its range r, its x, y, and z coordinates, and its remission, and we store them in the image, creating a $[5 \times h \times w]$ tensor. Because of the de-skewing of the scan, the assignment of each points to its corresponding (u, v) is done in a descending range order, to ensure that all points rendered in the image are in the current field of view of the sensor. We furthermore save this list of (u, v) pairs to gather and clean the semantics of the resulting point cloud, as we describe in Sec. III-C and Sec. III-D.

B. Fully Convolutional Semantic Segmentation

To obtain the semantic segmentation of this range image representation of the point cloud we use a 2D semantic segmentation CNN, which is modified to fit this particular input type and form factor. Similarly to Wu *et al.* [21], we use an encoder-decoder hour-glass-shaped architecture, which is depicted in Fig. 3. These types of deep hour-glass-shaped segmentation networks are characterized by having an encoder with significant downsampling, which allows the higher abstraction deep kernels to encode context information, while running faster than non-downsampling counterparts. In our case, this downsampling is 32 (see Fig. 3). This is later followed by a decoder module which upsamples the "feature code" extracted by the convolutional backbone

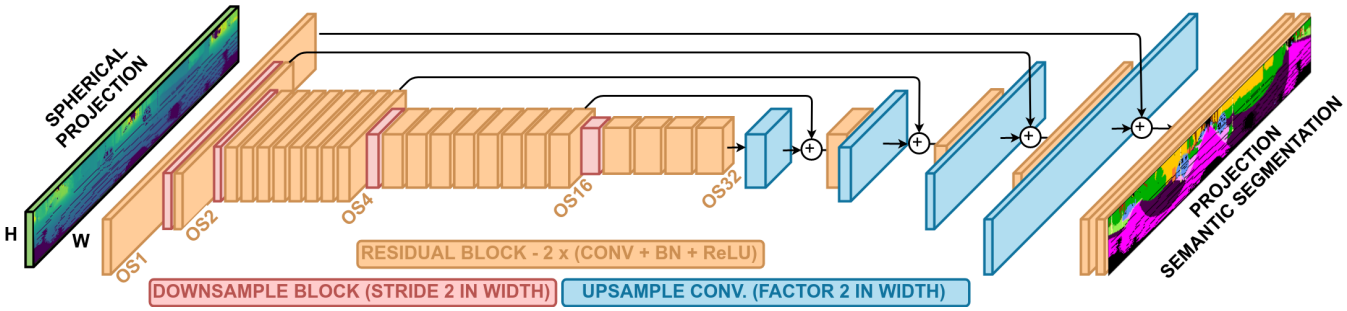

Fig. 3: Our fully convolutional semantic segmentation architecture. RangeNet53 is inspired in a Darknet53 Backbone [16].

encoder to the original image resolution, adding also convolutional layers to refine these results. At the same time, after each upsampling we also add skip connections between different levels of output stride (OS) of the encoder and sum them to the corresponding output stride feature volume in the decoder, illustrated by the black arrows, to recover some of the high-frequency edge information that gets lost during the downsampling process. After this encoding-decoding behavior, the last layer of the architecture performs a set of $[1 \times 1]$ convolutions. This generates an output volume of $[n \times h \times w]$ logits, where n is the number of classes in our data. The last layer during inference is a softmax function over the unbounded logits of the form $\hat{y}_c = \frac{e^{\text{logit}_c}}{\sum_c e^{\text{logit}_c}}$. This gives a probability distribution per pixel in the range image, where logit_c is the unbounded output in the slice corresponding to class c. During training, this network is optimized end to end using stochastic gradient descent and a weighted cross-entropy loss $\mathcal{L}$:

$$\mathcal{L} = -\sum_{c=1}^{C} w_c \, y_c \, \log\left(\hat{y}_c\right), \text{ where } w_c = \frac{1}{\log\left(f_c + \epsilon\right)} \quad (2)$$

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

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

C. Point Cloud Reconstruction from Range Image

The common practice to map from a range image representation to a point cloud is to use the range information, along with the pixel coordinates and the sensor intrinsic calibration to realize a mapping $\Pi^* : \mathbb{R}^2 \mapsto \mathbb{R}^3$. However, since we are generating the range image from a point cloud originally, as stated in Sec. III-A, this could mean dropping a significant number of points from the original representation. This is especially critical when using smaller images in order to make the inference of the CNN faster. E.g. a scan with 130 000 points projected to a $[64 \times 512]$ range image will represent only 32 768 points, sampling the closest point in each pixel's frustum. Therefore, to infer all the original points in the semantic cloud representation, we use all the (u, v) pairs for all the $\mathbf{p}_i$ obtained during the initial rendering process and index the range image with the image coordinates that correspond to each point. This can be performed extremely fast in the GPU before the next post processing step takes place, and it results in a semantic label for each point that was present in the entire input scan, in a loss-less way.

D. Efficient Point Cloud Post-processing

Unfortunately, the benefits of the expedite semantic segmentation of LiDAR scans through 2D semantic segmentation of range images does not come without draw-backs. The encoder-decoder hour-glass-like CNNs provide blurry outputs during inference, which is also a problem for RGB and RGBD semantic segmentation. Some methods, such as [21] use a conditional random field over the predictions in the image domain after the 2D segmentation to eliminate this "bleeding" of the output labels. Using the softmax probabilities of each pixel as unary potentials for the CRF, and penalizing jumps in signal and Euclidean distance between neighboring points. Even though this helps in 2D, it does not fix the problem after the re-projection to three-dimensional space, since once the labels are projected into the original

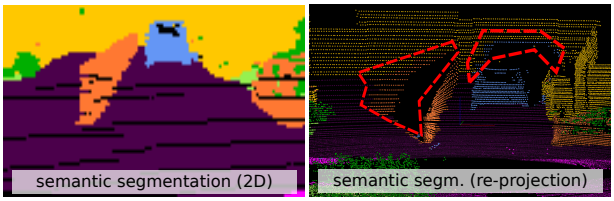

Fig. 4: Illustration of the label re-projection problem. Both the fence and the car in the range image (left) were given the proper semantic label, but during the process of sending the semantics back to the original points (right), the labels were also projected as "shadows".

point cloud, two or more points which were stored into the same range image pixel will get the same semantic label. This effect is illustrated in Fig. 4, where the labels of the inferred point cloud present shadows in objects in the background due to the blurry CNN mask, and the mentioned discretization. Moreover, if we wish to use smaller range image representations to infer the semantics, this problem becomes even stronger, resulting in shadow-like artifacts of the semantic information in objects of different classes.

To solve this problem, we propose a fast, GPU-enabled, k-nearest neighbor (kNN) search operating directly in the input point cloud. This allows us to find, for each point in the semantic point cloud, a consensus vote of the k points in the scan that are the closest to it in 3D. As it is common in kNN search, we also set a threshold for the search, which we call *cut-off*, setting the maximum allowed distance of a point considered a near neighbor. The distance metric to rank the k closest points can be the absolute differences in the range, or the Euclidean distance. Although, we also tried to use the remission as a penalization term, which did not help in our experience. From now on, we explain the algorithm considering the usage of the absolute range difference as the distance, but the Euclidean distance works analogously, albeit being slower to compute.

We explain the steps of our algorithm, described in Alg. 1, referring to the corresponding line numbers. Our approximate nearest neighbor search uses the range image representation to obtain, for each point in the $[h, w]$ range image, an $[S, S]$ window around it in the image representation, with S being a value found empirically through a grid-search in the validation set. This operation is performed through the *"im2col"* primitive, which is internally used by most parallel computing libraries to calculate a convolution, and therefore directly accessible through all deep learning frameworks. This results in a matrix of dimension $[S^2, hw]$, which contains an unwrapped version of the $[S, S]$ neighborhood in each column, and each column center contains the actual pixel's range (lines 2–4). As not all points are represented in the range image, we use the (u, v) tuples for each $\mathbf{p}_i$ obtained during the range image rendering process, and extend this representation to a matrix of dimension $[S^2, N]$, containing the range neighborhoods of all the scan points (lines 5–7). As this is done by indexing the unfolded image matrix, the centers of columns don't represent the actual range values anymore. Thus, we replace the center row of the matrix by the actual range readings for each point. The result of this is a $[S^2, N]$ matrix which contains all the range readings for the points in the center row, and in each column, its unwrapped $[S, S]$ neighborhood (lines 8–9). This is a key checkpoint in the algorithm, because it allows us to find in a quick manner a set of S^2 candidates to consider during the neighbor search for each point, in parallel. This allows our algorithm to run orders of magnitude faster than other nearest neighbor search approaches such the ones in FLANN, which work in unordered point clouds, by exploiting the arrangement of the scan points in the sensor. This key structural difference allows us to run in real-time even for large point clouds.

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

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

After this, we need to find the k closest points for each column containing the S^2 candidates, which is a done through an *argmin* operation (lines 28–29). This allows us to get the indexes for the k points in the S neighborhood with the least weighted distance.

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

It is important to notice that, given the independence of all the points inside the loops in Alg. 1, each of the main components can be represented either with a parallel computing primitive or in a highly vectorized way, both

Algorithm 1: Efficient Projective Nearest Neighbor Search for Point Labels

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

Let be $[l{:}u] = \{i \mid l \leq i \leq u\}$ the range from l to u.
/* Get S^2 neighbors $\mathbf{N}'$ for each pixel */
foreach $(u, v) \in [1{:}W] \times [1{:}H]$ **do**
  foreach $(i, j) \in [1{:}S] \times [1{:}S]$ **do**
    $\mathbf{N}'[v \cdot W + u, j \cdot S + i] = \mathbf{I}_{\text{range}}[u + i, v + j]$
/* Get neighbors $\mathbf{N}$ for each point */
foreach $(u, v) \in \mathcal{C}$ **do**
  foreach $(i, j) \in [1{:}S] \times [1{:}S]$ **do**
    $\mathbf{N}[v \cdot W + u, i \cdot S + j] = \mathbf{N}'[v \cdot W + u, i \cdot S + j]$
/* Fill in real point ranges */
foreach $i \in [1{:}N]$ **do**
  $\mathbf{N}[i, \lfloor (S \cdot S - 1)/2 \rfloor] = \mathcal{R}(i)$
/* Label neighbors $\mathbf{L}'$ for each pixel */
foreach $u \in [1{:}W]$, $v \in [1{:}H]$ **do**
  foreach $(i, j) \in [1{:}S] \times [1{:}S]$ **do**
    $\mathbf{L}'[v \cdot W + u, i \cdot S + j] = \mathbf{I}_{\text{label}}[u + i, v + j]$
/* Get label neighbors $\mathbf{L}$ for each point */
foreach $(u, v) \in \mathcal{C}$ **do**
  foreach $(i, j) \in [1{:}S] \times [1{:}S]$ **do**
    $\mathbf{L}[v \cdot W + u, i \cdot S + j] = \mathbf{L}'[v \cdot W + u, i \cdot S + j]$
/* Distances to neighbors $\mathbf{D}$ for each point */
foreach $i \in n\,[1{:}N]$ **do**
  foreach $j \in n\,[1{:}S{\cdot}S]$ **do**
    $\mathbf{D}[i, j] = |\mathbf{N}[i, j] - \mathcal{R}(i)|$
/* Compute inverse Gaussian Kernel */
Let $\mathcal{N}(u \mid \mu, \sigma)$ be a Gaussian with mean μ and std. deviation σ.
foreach $(i, j) \in [1{:}S] \times [1{:}S]$ **do**
  $\mathbf{G}'[j \cdot S + i] = \mathcal{N}(i \mid 0, \sigma) \cdot \mathcal{N}(j \mid 0, \sigma)$
Let be $G_{\max} = \max\{\mathbf{G}'[i] \mid i \in [1{:}S{\cdot}S]\}$ the maximium of $\mathbf{G}'$
foreach $i \in [1{:}S{\cdot}S]$ **do**
  $\mathbf{G}[i] = 1 - G_{\max} \cdot \mathbf{G}'[i]$
/* Weight neighbors with inverse Gaussian kernel */
foreach $i \in [1{:}N]$ **do**
  foreach $j \in [1{:}S{\cdot}S]$ **do**
    $\mathbf{D}[i, j] = \mathbf{D}[i, j] \cdot \mathbf{G}[j]$
/* Find k-nearest neighbors $\mathcal{S}$ for each point */
foreach $i \in [1{:}N]$ **do**
  $\mathcal{S}[i] = \{j \mid |\{n \in [1{:}S{\cdot}S] \mid \mathbf{D}[i, n] < \mathbf{D}[i, j]\}| \leq k\}$
/* Gather votes. */
foreach $i \in [1{:}N]$ **do**
  $n = 1$
  foreach $j \in \mathcal{S}[i]$ **do**
    if $D[i, j] > \delta_{\textit{cutoff}}$ **then**
      $\mathbf{L}_{\text{knn}}[i, n] = C + 1$
    else
      $\mathbf{L}_{\text{knn}}[i, n] = \mathbf{L}[i, j]$
    $n = n + 1$
/* Accumulate votes. */
foreach $i \in [1{:}N]$ **do**
  foreach $j \in [[:1] : k]$ **do**
    if $\mathbf{L}_{\textit{knn}}[i, j] \leq C$ **then**
      $\mathbf{V}[i, \mathbf{L}_{\text{knn}}[i, j]] = \mathbf{V}[i, \mathbf{L}_{\text{knn}}[i, j]] + 1$
/* Find maximum consensus. */
foreach $i \in [1{:}N]$ **do**
  $\mathbf{L}_{\text{consensus}}[i] = \arg\max_c \mathbf{L}_{\text{knn}}[i, c]$

of which are directly implementable in a GPU, using off-the-shelf, high abstraction deep learning or data science frameworks.

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

IV. Experimental Evaluation

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

Dataset. We train and evaluate our approach on a large-scale dataset that provides dense point-wise annotations for the entire KITTI Odometry Benchmark [7], [1]. The dataset is comprised of over $43\,000$ scans from which over $21\,000$ from sequences 00 to 10 are available for training and the remaining scans from sequences 11 to 21 are used as test set. We used sequence 08 as validation set for hyperparameter selection and trained our approach on the remaining training sequences. Overall, the dataset provides 22 classes from which 19 classes are evaluated on the test set via our benchmark website.

Hyperparameter selection. All hyperparameters for RangeNet models are selected and evaluated on the validation set (sequence 8). For all backbone trainings, we use a learning rate of $1 \cdot 10^{-3}$, with a decay of 0.99 every epoch, and train for 150 epochs. For all CNN backbones, convergence was achieved in less than 150 epochs. For all the state-of-the-art methods, the hyperparameters were also selected on the validation set.

Metrics. To assess the labeling performance, we use the commonly applied mean Jaccard Index or mean intersection-over-union (IoU) metric, mIoU, over all classes [6] given by

$$mIoU = \frac{1}{C} \sum_{c=1}^{C} \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c + \text{FN}_c} \tag{3}$$

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

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

TABLE I: IoU [%] on test set (sequences 11 to 21). RangeNet21 and RangeNet53 represent the new baselines with augmented Darknet backbones (21 and 53 respectively), and the versions with (++) are treated with our fast point cloud post-processing based on range.

Approach	Size	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign	mean IoU	Scans/sec
Pointnet [14]	50000pts	46.3	1.3	0.3	0.1	0.8	0.2	0.2	0.0	61.6	15.8	35.7	1.4	41.4	12.9	31.0	4.6	17.6	2.4	3.7	14.6	2
Pointnet++ [15]		53.7	1.9	0.2	0.9	0.2	0.9	1.0	0.0	72.0	18.7	41.8	5.6	62.3	16.9	46.5	13.8	30.0	6.0	8.9	20.1	0.1
SPGraph [10]		68.3	0.9	4.5	0.9	0.8	1.0	6.0	0.0	49.5	1.7	24.2	0.3	68.2	22.5	59.2	27.2	17.0	18.3	10.5	20.0	0.2
SPLATNet [19]		66.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	70.4	0.8	41.5	0.0	68.7	27.8	72.3	35.9	35.8	13.8	0.0	22.8	1
TangentConv [20]		86.8	1.3	12.7	11.6	10.2	17.1	20.2	0.5	82.9	15.2	61.7	9.0	82.8	44.2	75.5	42.5	55.5	30.2	22.2	35.9	0.3
SqueezeSeg [21]	64 × 2048 px	68.8	16.0	4.1	3.3	3.6	12.9	13.1	0.9	85.4	26.9	54.3	4.5	57.4	29.0	60.0	24.3	53.7	17.5	24.5	29.5	**66**
SqueezeSeg-CRF [21]		68.3	18.1	5.1	4.1	4.8	16.5	17.3	1.2	84.9	28.4	54.7	04.6	61.5	29.2	59.6	25.5	54.7	11.2	36.3	30.8	55
SqueezeSegV2 [22]		81.8	18.5	17.9	13.4	14.0	20.1	25.1	3.9	88.6	45.8	67.6	17.7	73.7	41.1	71.8	35.8	60.2	20.2	36.3	39.7	50
SqueezeSegV2-CRF [22]		82.7	21.0	22.6	14.5	15.9	20.2	24.3	2.9	88.5	42.4	65.5	18.7	73.8	41.0	68.5	36.9	58.9	12.9	41.0	39.6	40
RangeNet21 [Ours]		85.4	26.2	26.5	18.6	15.6	31.8	33.6	4.0	91.4	57.0	74.0	26.4	81.9	52.3	77.6	48.4	63.6	36.0	50.0	47.4	20
RangeNet53 [Ours]	64 × 2048 px	86.4	24.5	32.7	25.5	22.6	36.2	33.6	4.7	**91.8**	64.8	74.6	**27.9**	84.1	55.0	78.3	50.1	64.0	38.9	52.2	49.9	13
	64 × 1024 px	84.6	20.0	25.3	24.8	17.3	27.5	27.7	7.1	90.4	51.8	72.1	22.8	80.4	50.0	75.1	46.0	62.7	33.4	43.4	45.4	25
	64 × 512 px	81.0	9.9	11.7	19.3	7.9	16.8	25.8	2.5	90.1	49.9	69.4	2.0	76.0	45.5	74.2	38.8	62.7	25.5	38.1	39.3	52
RangeNet53++ [Ours+kNN]	64 × 2048 px	**91.4**	**25.7**	**34.4**	**25.7**	**23.0**	**38.3**	**38.8**	**4.8**	**91.8**	**65.0**	**75.2**	27.8	**87.4**	**58.6**	**80.5**	**55.1**	**64.6**	**47.9**	**55.9**	**52.2**	12
	64 × 1024 px	90.3	20.6	27.1	25.2	17.6	29.6	34.2	7.1	90.4	52.3	72.7	22.8	83.9	53.3	77.7	52.5	63.7	43.8	47.2	48.0	21
	64 × 512 px	87.4	9.9	12.4	19.6	7.9	18.1	29.5	2.5	90.0	50.7	70.0	2.0	80.2	48.9	77.1	45.7	64.1	37.1	42.0	41.9	38

A. *Performance of RangeNet++ w.r.t. State-of-the-art*

The first experiment is designed to support our claim that our approach over-performs the state of the art in the task of scene semantic segmentation of LiDAR point clouds. Tab. I shows the difference between our RangeNet backbones, using 21 and 53 layers, and 7 other baseline methods. The superior performance of our RangeNet baselines, even without our cleaning, for all the input resolutions of shows that it is a solid baseline to benchmark our efficient kNN cleaning. Tab. I also shows that our method, RangeNet++, which includes our kNN post processing consistently beats its unprocessed RangeNet counterpart, showing the efficacy of our kNN search. The kNN cleaning is consistently better for all but one classes, unlike the CRF, which is a conclusion reached by the original SqueezeSeg [21] paper as well, even when the overall IoU is higher.

B. *Ablation Studies*

The second experiment shows the influence of the k and S parameters in the validation set. For each of the 4 parameters k, S, σ, and *cut-off* we chose a wide range of values and evaluated the result of post-processing the inference results of the RangeNet53 backbones for all input resolutions. Fig. 5 shows a normalized result of the IoU in the validation set for each parameter set, for various k and S and the argmax of σ and *cut-off*. The results also show that we can obtain similar results using small kernels and the absolute range difference, as a proxy for the Euclidean distance. This supports our statement that this range difference is a good proxy for the actual distance the closer the points are in the image.

C. *Post-Processing Influence*

The third experiment is designed to support our claim that our algorithm improves the reconstruction of the semantics of the entire point cloud even for smaller range image resolutions. For this, we use our *border-IoU* metric, which only considers points that are a certain number of points away from a change in label. In Fig. 6 we show

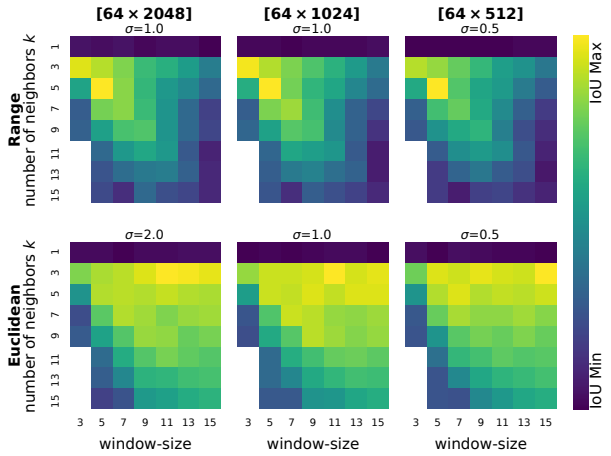

Fig. 5: Hyperparameter search post-processing for both Range (top row) and Euclidean distance (bottom row) using RangeNet53++, and different input resolutions. All experiments used *cutoff* = 1.0 m.

the value of the IoU and the value of the border IoU for different distances to border. Note that our post-processing approach doesn't only improve the IoU score by a couple of % points, but it significantly improves the border IoU score for low values of the distance to border parameter. This means that our approach is especially useful to help in cases of label "bleeding" or "shadowing" described in Sec. III-D. Another important conclusion is that there is only marginal differences between using the faster to compute range difference and the actual Euclidean distance, throughout the entire spectrum of border distances, and in the IoU, which support our statement that it is a good approximation.

D. *Runtime*

The fourth experiment is designed to support our claim that the approach can run in its totality online in a moving platform, using a single GPU. Tab. II shows the runtime for the backbone, different post-processing distance functions (for the best parameters), and the total time required. As expected, the range-based post-processing is faster to calcu-

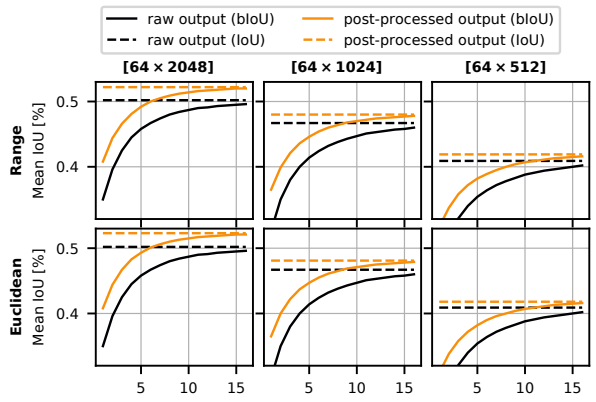

Fig. 6: Border IoU (bIoU) and IoU as a function of the distance to label change. This plot shows that our post-processing improves the IoU, and significantly improves the borderIoU, which means that it recovers blurry mask and discretization errors better.

TABLE II: Runtime of RangeNet53++.

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

late, since each distance calculation requires a subtraction and an absolute value, compared to 3 squares, 2 sums, and a square root. Therefore, since the difference in performance is negligible, we use the sum of our CNN backbone plus this range processing time for our total runtime, which we evaluate in two different types of hardware.

V. Conclusion

In this work, we presented a fast and accurate framework for semantic segmentation of point clouds recorded by a rotating LiDAR sensor. Our main contribution is a novel deep-learning-supported approach that exploits range images and 2D convolutions, followed by a novel, GPU-accelerated post-processing to recover consistent semantic information during inference for entire LiDAR scans. Our experimental evaluation suggests that our modified 2D deep CNN operating on range images outperforms the current state of the art in semantic segmentation of LiDAR point clouds. Moreover, our efficient, GPU-enabled post-processing can further improve on these results by recovering important boundary information lost during the de-skewing of the laser scans, the lossy discretization into a proxy representation, and the inference through an hour-glass-shaped CNN. Overall, our approach outperforms the state of the art both in accuracy and runtime, taking a step forward towards sensor redundancy for semantic segmentation for autonomous vehicles and robots.

References

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

[2] G.J. Brostow, J. Fauqueur, and R. Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 2008.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[21] B. Wu, A. Wan, X. Yue, and K. Keutzer. SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[22] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer. SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud. *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[23] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. *arXiv preprint*, abs/1612.01105, 2016.