LiDAR-based Panoptic Segmentation via Dynamic Shifting Network
NPM3D - Project Report

Alexis GROSHENRY - alexis.groshenry@polytechnique.edu

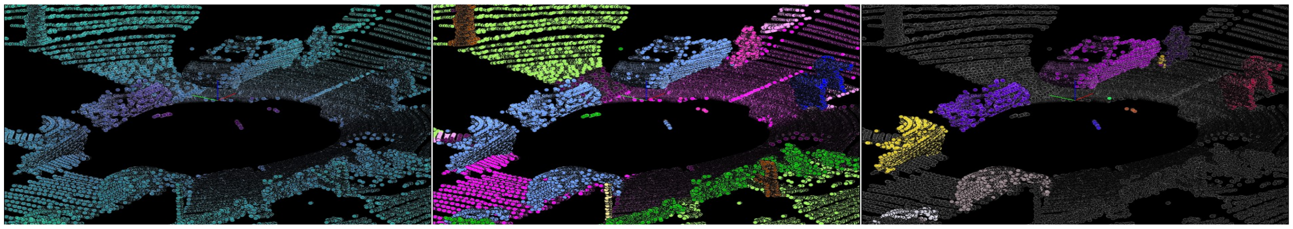
Figure 1. DS-Net results on SemanticKITTI. From left to right : LiDAR point cloud, ground-truth segmentation, predicted segmentation

Abstract

This project report studies the method DS-Net for panoptic segmentation on LiDAR point clouds, which was introduced in [8]. At the time the article was published, DS-Net ranked first on the SemanticKITTI leaderboard, one of the main dataset to test panoptic segmentation models in the context of autonomous driving. The proposed approach adapts several methods for semantic and instance segmentation, and combines them to propose a global pipeline solving the panoptic segmentation task and overcoming the challenging setting of LiDAR point clouds. After a short presentation of the method, the results obtained with an implementation adapted from an open-source codebase are analyzed, and several improvements of DS-Net are also proposed to enhance its performance, while keeping the main ideas of the original approach.

1. Introduction

In the context of autonomous driving, the acquisition and exploitation of information from the environment of the car is crucial. A number of methods address this challenging task by performing segmentation of the point cloud acquired by the car's sensors. This segmentation step requires to perform two closely related but yet different tasks :

- **Semantic segmentation** classifies the different points of the cloud into separate classes
- **Instance segmentation** distinguishes between separate objects of the same class.

The combination of these two complementary tasks is called panoptic segmentation and is a recent topic for point clouds, and especially LiDAR point clouds for which very few methods have been proposed specifically. Indeed, most methods proposed for indoor point clouds [5, 9] fail on outdoor LiDAR clouds because of their more challenging nature : the scenes are very heterogeneous with large scale gaps between instances, and the density of points varies a lot and heavily depends on the distance to the sensor.

Hong et al. propose in 2020 DS-Net[8] and claim to be one of the first attempts to address panoptic segmentation on LiDAR point clouds. At the time the paper was released, they ranked first on the *SemanticKITTI Panoptic Segmentation* competition. The general idea of the method is to extract features from a convenient representation of the point cloud overcoming the difficulties inherent to LiDAR point clouds, and to rely on these features to perform semantic segmentation, which is finally completed by instance segmentation realized by a class agnostic clustering algorithm.

2. DS-Net : the model

2.1. Overview of the architecture

Given the complexity of panoptic segmentation which requires the resolution of both LiDAR-based semantic and instance segmentations, the overall architecture can be seen as a stack of several modules (Fig. 2) either inspired from the literature, or entirely designed by the authors, and addressing different parts of the global problem. The following paragraphs describe these different modules in more detail.

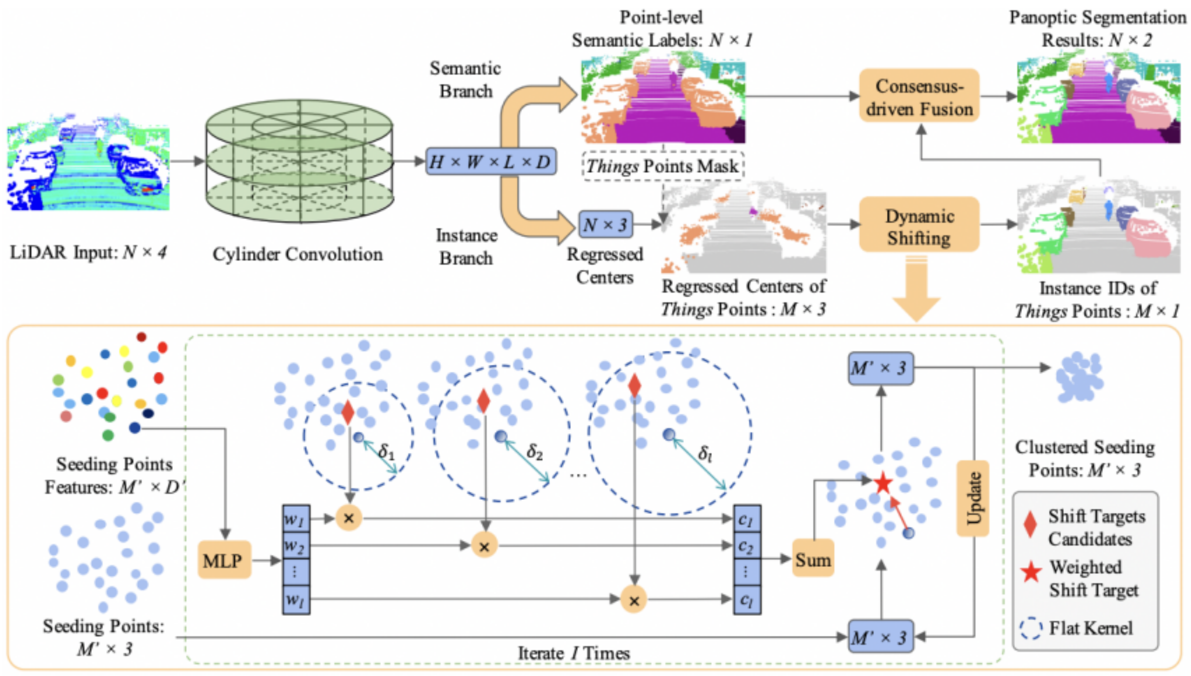
Point-level Semantic Labels: $N \times 1$
Panoptic Segmentation Results: $N \times 2$
Semantic Branch
Consensus-driven Fusion
$H \times W \times L \times D$
Things Points Mask
LiDAR Input: $N \times 4$
Cylinder Convolution
Instance Branch
$N \times 3$
Regressed Centers
Dynamic Shifting
Regressed Centers of Things Points : $M \times 3$
Instance IDs of Things Points : $M \times 1$
Seeding Points Features: $M' \times D'$
Seeding Points: $M' \times 3$
MLP
Sum
Update
$M' \times 3$
Clustered Seeding Points: $M' \times 3$
Shift Targets Candidates
Weighted Shift Target
Flat Kernel
Iterate I Times

Figure 2. Architecture of DS-Net - source [8]

2.2. Strong Backbone

Cylindrical Voxel Representation Since the density of points in the LiDAR point cloud depends a lot on the distance to the sensor, a regular voxel Cartesian partition is not adapted to represent the input point cloud. Inspired by Cylinder3D [14], the authors prefer a cylindrical voxel representation which allows to allocate voxels whose volumes are coherent with the density of points at the considered location : the closer the voxel is to the sensor, the smaller is the voxel, but this allow to get a more even distribution of point across the voxels of the grid.

The obtained cylinder partition is then processed by a PointNet architecture[12] to produce a set of geometric features for each voxel. A U-Net architecture is finally optimized to extract relevant features for the tasks performed by the other modules of the network, and in particular the semantic and instance branches.

Semantic Branch From the aforementioned shared features, a MLP computes semantic confidence scores for each voxel from the previous representation.

Instance Branch The features of the points classified as *things* by the Semantic Branch are further processed by a new MLP to predict the centers of their associated instances.

2.3. Dynamic-Shifting Module

The Dynamic-Shifting module realizes the clustering of the regressed centers of the instances and is the main contribution of the authors. It leverages ideas from Mean Shift[4], a clustering method robust to noise and density changes, which makes it particularly well adapted for LiDAR point clouds. We shortly describe here its main principles. For a subset of points $X \in \mathbb{R}^{M\times 3}$ sampled from the set of points to be clustered , a shift vector $S = f(X) - X$ is computed by a kernel function f and is then used to update the points to make them closer to the estimated positions of the cluster centers. $X \leftarrow X + S = f(X)$. This update step is then repeated for a fixed number of iterations, or until a convergence criterion is satisfied. From the shifted set of points, the cluster centers are extracted and all points to cluster are finally assigned to the closest cluster centers.

However, if this method performs well on indoor point clouds[11], it comes to fail when directly applied on outdoor point clouds because of important scale gaps between different instances (e.g. pedestrian vs truck). Indeed, the Mean Shift method requires the choice of a bandwidth hyperparameter in the design of the kernel function, which strongly affects the results of the clustering algorithm. This causes a drop in performance when both large and small instances need to be clustered at the same time.

In order to overcome this limitation, the authors introduce the Dynamic Shifting Module which essentially computes l values of the shift vector S for l candidate values of the bandwidth parameter. These bandwidth candidates chosen according to the geometric properties of the instances present in the data. Finally, a MLP computes a weight matrix $W \in \mathbb{R}^{M\times l}$ to weigh the actual contribution of each of the l shift vectors, in order to dynamically adapt towards the appropriate bandwidth. Like in the regular Mean Shift method, this procedure is repeated for a fixed number of iterations.

2.4. Consensus-driven Fusion

Finally, one needs to make the semantic and the instance predictions consistent with one another. Indeed, since the Dynamic-Shifting is done in a class agnostic way, points that have been classified in different classes by the semantic branch might however end up in the same cluster. In order to overcome these inconsistencies, the authors propose a simple but efficient procedure based on majority voting. For each instance computed by the Dynamic-Shifting module, the label of its points are overwritten with the majority label within the instance.

2.5. Network optimization

The optimization of the whole pipeline is done in a step by step fashion, each of the presented module being optimized one after another.

At first, following the protocol presented in Cylinder3D [14], the feature extractors of the **cylinder convolution** (PointNet and U-Net) and the **semantic branch** are trained using a combination of a cross entropy loss $L_{sem,acc}$ controlling the accuracy of the classification, and a lovasz-softmax loss [3] L_{IoU} which controls the IoU of the segmentation. For this step the total loss writes :

$$L_{sem} = L_{sem,acc} + L_{sem,IoU} \tag{1}$$

In a second step, the **instance branch** is trained to realize the regression of the centers of instance points. It is once again based on the features computed by the cylinder convolution, whose weights are frozen in this step. The learning objective is a simple L1 loss between the predicted center and the ground-truth center of the corresponding instance.

$$L_{inst} = \frac{1}{M} \sum_{x=1}^{M} \|P[x] + O[x] - C_{GT}[x]\|_1 \tag{2}$$

Finally, the **Dynamic Shifting module** is trained using again the sum over all iterations of a L1 loss between the predicted center at each iteration and the groundtruth center of the corresponding instance.

$$L_{DS} = \frac{1}{M} \sum_{i=1}^{I} \sum_{x=1}^{M} \|X_i[x] + S_i[X] - C_{GT}[x]\|_1 \tag{3}$$

3. Experiments

3.1. Implementation details

For my experiments, I mainly relied on the official implementation of the authors[1] and the provided pretrained checkpoints. The different architectures are coded using the open-scoure library spconv[2] which provides an optimized implementation of sparse convolutions. It is important to underline that I had quite a hard time trying to properly reproduce the environment and I notably had to manually modify CMake compilation scripts to solve incompatibilities within the environment. These problems were in general linked to the fact that the authors directly used the implementation of Cylinder3D[3] for cylinder convolution, which uses a deprecated version of spconv.

I also adapted some of the visualization scripts available here[4] to provide nice and useful visualizations of the semantic and instance predictions and compare it easily with the ground-truth labels. I also added a number of functions to interact dynamically with the visualization in order to navigate through the sequence of frames, highlight the misclassified points, change the size of points etc (see Fig. 3). These visualization functions were essential to propose a qualitative analysis of the segmentation results.

3.2. SemanticKITTI

I conducted my experiments on the SemanticKITTI dataset[6, 2], which is a large dataset of LiDAR point clouds corresponding to 360° views around a car. It is one of the main dataset for semantic understanding in the context of autonomous driving.

It contains 22 sequences, each corresponding to a different trajectory of a car, and is split between 10 sequences for training, 1 sequence for validation and 11 sequences for testing. For the test sequences, the ground-truth labels are not available since they are used to eventually rank the methods. There are 28 classes across the dataset, which are further separated between 19 *stuff* classes and 9 *things* classes on which instance segmentation is performed.

[1] https://github.com/hongfz16/DS-Net
[2] https://github.com/traveller59/spconv
[3] https://github.com/xinge008/Cylinder3D
[4] https://github.com/PRBonn/semantic-kitti-api

Model	PQ			SQ			RQ			IoU
	stuff	things	mean	stuff	things	mean	stuff	things	mean	mean
Backbone	0.5483	0.5856	0.5640	0.7703	0.7582	0.7652	0.6740	0.6671	0.6711	0.6349
DS-Net	0.5478	0.6180	0.5773	0.7705	0.7824	0.7755	0.6732	0.6882	0.6795	0.6348

Table 1. Backbone vs DS-Net performance

3.3. Results reproduction

At first, I wanted to retrain separately different modules of the model and check if the obtained performance were in accordance with the results reported in the original article for the SemanticKITTI dataset. I also wanted to play with the tuning of hyperparameters in order to study their influence on the performance of the model.

However, I had to renounce to these retrainings because a single training epoch would take more than 10 hours on my Google Cloud VM, and the authors did 50. The authors used 4 Tesla V100 GPUs parallelized and claimed that processing a single frame took 0.5 second. Considering the 20 000 training frames and ignoring the validation epochs, the training of one part of the network took them 5 days! For this reason, I preferred to stay with a K80 GPU and focus on testing experiments, rather than training a single module of DS-Net, which would have given similar results than the authors.

Using the pretrained checkpoints provided by the authors, I evaluated the model on the validation frames of the SemanticKITTI dataset, which are the frames from the sequence 08. The inference on this sequence took a bit longer than 2 hours, which is a reasonable time to conduct experiments, but is actually slow in the context of autonomous cars where the point clouds need to be processed in real time. The table 3.2 compares the backbone model with the complete DS-Net by computing the same metrics as in the original paper : the IoU, the panoptic quality *PQ*, the segmentation quality *SQ* and the recognition quality *RQ*. The metrics are computed for the *stuff* classes, the *things* classes and finally all classes.

We observe that adding the Dynamic-Shifting module to the backbone model preserves the performance on the *stuff* classes, but brings a significant increase in performance on *things* classes for all performances. This increase is such that the performance of the complete DS-Net is better on the *things* classes than on the stuff ones, while this was true only for the panoptic quality metric for the backbone model.

Diving into the performance per class with the table 3.3, we start with a few general observations :

- the classes *motorcyclist* and *other-ground* get very low scores because they are almost not present in the validation sequence 08
- apart from these two classes, the classes *truck*, *parking* and *fence* get particularly low Panoptic Quality scores with the backbone model

- in the context of autonomous driving, the model gets good scores for other users on the road (*car*, *person*, *bicyclist*, *motorcyclist*) and for some important elements of its environment (*road*, *sidewalk*, *building*), its performance on some other classes stay poor even if they are particularly important for the understanding of the relevant environment of the car (*trunk*, *parking*, *pole*, *traffic-sign*)

We observe that the Dynami-Shifting module mostly improves the performance of the class **truck**, its panoptic quality being increased by almost 25 points to eventually reach a very good score. The predictions on some other classes are also improved by a few points but those are details compared to the improvements on the *truck* class. However, there is no gain in performance for the two other problematic classes *fence* and *parking*.

Class	IoU		PQ		RQ		SQ	
	Backbone	DS-Net	Backbone	DS-Net	Backbone	DS-Net	Backbone	DS-Net
car	0.9559	0.9541	0.9182	0.9259	0.9850	0.9856	0.9322	0.9394
bicycle	0.4782	0.4773	0.5412	0.5253	0.7056	0.6888	0.7670	0.7626
motorcycle	0.6044	0.5812	0.6177	0.6040	0.6885	0.6757	0.8971	0.8940
truck	**0.7805**	**0.8151**	**0.3848**	**0.6316**	**0.5122**	**0.6777**	**0.7514**	**0.9319**
other-vehicle	0.6006	0.5892	0.5585	0.5877	0.6286	0.6455	0.8885	0.9105
person	0.6986	0.7002	0.7648	0.7739	0.8516	0.8681	0.8980	0.8914
bicyclist	0.8412	0.8474	0.8995	0.8959	0.9655	0.9639	0.9317	0.9295
motorcyclist	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
road	0.9344	0.9346	0.9351	0.9353	0.9995	0.9995	0.9355	0.9357
parking	0.4118	0.4125	0.1369	0.1387	0.1951	0.1973	0.7020	0.7030
sidewalk	0.7904	0.7910	0.7580	0.7587	0.9194	0.9202	0.8244	0.8245
other-ground	0.0415	0.0414	0.0048	0.0049	0.0086	0.0086	0.5628	0.5642
building	0.9122	0.9116	0.8819	0.8818	0.9726	0.9729	0.9067	0.9064
fence	0.6167	0.6145	0.2140	0.2129	0.3009	0.2996	0.7111	0.7105
vegetation	0.8637	0.8636	0.8488	0.8478	0.9818	0.9801	0.8645	0.8650
trunk	0.6960	0.6933	0.5346	0.5334	0.7184	0.7181	0.7441	0.7428
terrain	0.7024	0.7022	0.5611	0.5588	0.7652	0.7607	0.7333	0.7346
pole	0.6295	0.6276	0.5672	0.5638	0.7710	0.7658	0.7357	0.7362
traffic-sign	0.5043	0.5045	0.5891	0.5894	0.7822	0.7829	0.7531	0.7528
all	0.6349	0.6348	0.5640	0.5773	0.6711	0.6795	0.7652	0.7755

Table 2. Backbone vs DS-Net performance per class

3.4. Qualitative analysis

3.4.1 General observations

Using the implemented visualization functions, we observe that the segmentation results are pretty good on the whole. Having a closer look at the difference map between the ground-truth and the predictions shows that there are only a few points sparsely distributed over the whole image that are wrongly predicted (see Fig. 4). This overall satisfying quality should be enough in most situations to understand the environment of the car.

However, in some cases we observe large misclassified areas, where the model completely fails to correctly segment a whole set of points. This often happens between different categories of ground (parking, terrain, sidewalk), as in Fig. 7 where a parking area is classified as terrain and vegetation.

It is also interesting to look at the quality of segmentation against the distance to the sensor. This is done on Fig. 8 for all classes with a sufficient number of points in sequence 08, in order to avoid biases. It appears that for many classes the classification of points is the best at mid-distance, and gets particularly bad for low and large distances to the sensor. For short distances, this can be explained by the fact that a significant number of points are in the blind spot of the sensor, and thus the features computed do not encode the appropriate characteristics of the object. For large distances, small structures that are meaningful to distinguish between two classes with similar characteristics are erased by the distance to the sensor, explaining the observed drop in performance.

3.4.2 Errors in the context of autonomous driving

In the context of autonomous driving, some of the previous slight errors turn out to be problematic. For instance, in Fig. 5 the model makes confusion between the sidewalk and the road, which is very dangerous since this bad segmentation could make the car ride on the sidewalk. In addition to these semantic segmentation errors, instance segmentation false positive or false negative detections can have a heavy impact on the behaviour of the car. If there is a FP, then the car detects an instance where there is nothing, and it may trigger dangerous behaviours if the driving module tries to avoid a collision with this false instance. Similarly, a FN will lead to a wrong analysis of the environment of the car, which may cause a collision or delay the detection of a danger.

A recurrent situation across the validation frames is the case where another user of the road (car, bicyclist, motorcyclist) is wrongly classified when entering the blind spot around the LiDAR sensor (see Fig. 8). Since fewer points from the corresponding instance are detected, this makes the semantic classification harder, while this area is critical in the context of autonomous driving : if a bicycle or a car is right next to the car, this information is crucial for the autonomous car to adapt its behaviour. Even if this error occurs on only two or three consecutive frames, the sampling frequency being of 10Hz, these tenths of a second may be sufficient for the autonomous car to make a bad decision and cause an accident.

4. Possible improvements

Since the topic of panoptic segmentation on LiDAR point clouds is recent, there is room from improvements which could be achieved at different levels. To that end, I propose in this section several modifications of DS-Net to improve the results, without totally modifying the approach. Other approaches are of course promising given that DS-Net no longer ranks first on the SemanticKITTI leaderboard today, but I will not cover them in the following.

4.1. Towards a joint optimization of DS-Net

A first possible improvement concerns the general optimization of DS-Net. Indeed, it consists in a stack of many networks (5 different MLPs in total) which are trained and optimized separately using different learning objectives. In particular, the networks extracting the features from the point cloud are only trained alongside the semantic branch, and their weights are then fixed. As a consequence, these features are not optimized for the tasks of the instance branch and the Dynamic-Shifting module, while they are used as input features. This may hinder the final performance of these branches.

A solution to make these features better suited for every tasks they are used for, would be to conduct a single training on all parts of the network. The 3 losses presented in Sec. 2.5 would be backpropagated to the networks of DS-Net that are involved in the task associated to each loss. The detail of the losses required to update each part of DS-Net is summarized in Tab. 4.1.

	L_{sem}	L_{inst}	L_{DS}
Cylinder representation	X	X	X
Semantic Branch	X		
Instance Branch		X	X
Dynamic Shifting			X

Table 3. Joint optimization for multiple objectives

How to realize such a joint optimization is not straight forward since the different losses need to be scaled in order to prevent the optimization to focus on only one task. Moreover, one could also be facing memory issues since all gradients need to be stored, or gradient vanishing issues since the stacking of these multiple networks make the total network eventually very deep, in particular to backpropagate the loss associated to the Dynamic Shifting module. However, since the features computed by the cylinder representation module are used as inputs for the Dynamic Shifting, it acts as a residual connection like in very deep ResNet architectures [7], and should thus prevent gradient vanishing.

Another procedure using the same ideas, but which could ease the training, would be to alternate the training phases for the different tasks : train for N_{sem} epochs the semantic branch, for N_{inst} epochs the instance branch, and finally for N_{DS} epochs the Dynamic Shifting, and then loop on this procedure.

The joint optimization proposed here would require some work to balance the losses, choose the learning rates, or tune the number of epochs $N_{\{stage\}}$ aforementioned if the second option is chosen, but it is very likely that this work would improve the final performance of the global model.

4.2. Discussion about the instance head

It should be interesting to see if the instance head could be deleted, in which case the Dynamic-Shifting module would directly use the coordinates of the original points, instead of the regressed center. Adding more bandwidth candidates and potentially increasing the number of iterations in the algorithm could be sufficient to replace the work done beforehand by the instance head.

Such a modification would reduce the number of trainable parameters, speeding up and easing a potential joint optimization as suggested in Sec. 4.1. This would also reduce the inference time, which needs to be as fast as possible in the context of autonomous driving.

4.3. Temporal approach

Since the point clouds of the SemanticKITTI dataset are frames from a video sequence, it could be interesting to use the temporal dimension to improve the segmentation of a given frame. This is relevant in the context of autonomous driving, since one could use the T previous frames at time $t-1, t-2, ..., t-T$, to enhance the segmentation of the frame at time t.

A first possible way, would be to add a final post-processing step to make the segmentation consistent across frames. This could help to correct wrong predictions, for example by leveraging an easy prediction at mid-distance to improve predictions near the blind spot of the sensor. This can be easily implemented for instance segmentation by ensuring that overlapping instances across frames have the same label. This heuristic remains valid as long as the displacement of any instance between two frames (separated by 0.1 second) is shorter than its size. This would correspond to a maximum speed of $40m.s^{-1} = 144km.h^{-1}$ for a 4 meter long car, so it is a reasonable assumption.

Recently, several methods have been proposed to realize object tracking or panoptic segmentation on videos [10, 1, 13]. It could be possible to inspire from these methods to adapt the DS-Net architecture to take in input a temporal stack of 3D frames and to output the segmentation of the last frame. For example, we could modify the semantic and instance branches to take in input the concatenation of the semantic features of a temporal stack of frames. The modification would come at low cost since we could keep the weights of the cylinder representation network and simply modify the input dimension of the networks of the semantic and instances branches.

5. Conclusion

To conclude, in this project I studied the approach proposed by Hong et al. [8] for panoptic segmentation on LiDAR point clouds. If I was not able to retrain a part of the model due to a lack of computational resources, I still managed to test the method on the SemanticKITTI dataset and to analyze both numerically and visually the results, notably by implementing appropriate visualization functions. Since the topic is recent there is room for improvements, as suggested in the last section, but DS-Net is still a first very strong baseline for further approaches on the task of panoptic segmentation for LiDAR point clouds.

References

[1] Mehmet Aygün, Aljosa Osep, Mark Weber, Maxim Maximov, Cyrill Stachniss, Jens Behley, and Laura Leal-Taixé. 4d panoptic lidar segmentation. *CoRR*, abs/2102.12472, 2021. 7

[2] 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. 3

[3] Maxim Berman and Matthew B. Blaschko. Optimization of the jaccard index for image segmentation with the lovász hinge. *CoRR*, abs/1705.08790, 2017. 3

[4] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002. 2

[5] Francis Engelmann, Martin Bokeloh, Alireza Fathi, Bastian Leibe, and Matthias Nießner. 3d-mpa: Multi proposal aggregation for 3d semantic instance segmentation. *CoRR*, abs/2003.13867, 2020. 1

[6] 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. 3

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 6

[8] Fangzhou Hong, Hui Zhou, Xinge Zhu, Hongsheng Li, and Ziwei Liu. Lidar-based panoptic segmentation via dynamic shifting network. *CoRR*, abs/2011.11964, 2020. 1, 2, 7

[9] Li Jiang, Hengshuang Zhao, Shaoshuai Shi, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Pointgroup: Dual-set point grouping for 3d instance segmentation. *CoRR*, abs/2004.01658, 2020. 1

[10] Dahun Kim, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Video panoptic segmentation. *CoRR*, abs/2006.11339, 2020. 7

[11] Jean Lahoud, Bernard Ghanem, Marc Pollefeys, and Martin R. Oswald. 3d instance segmentation via multi-task metric learning. *CoRR*, abs/1906.08650, 2019. 2

[12] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017. 2

[13] Xinshuo Weng and Kris Kitani. A baseline for 3d multi-object tracking. *CoRR*, abs/1907.03961, 2019. 7

[14] Hui Zhou, Xinge Zhu, Xiao Song, Yuexin Ma, Zhe Wang, Hongsheng Li, and Dahua Lin. Cylinder3d: An effective 3d framework for driving-scene lidar semantic segmentation. *CoRR*, abs/2008.01550, 2020. 2, 3

Appendix

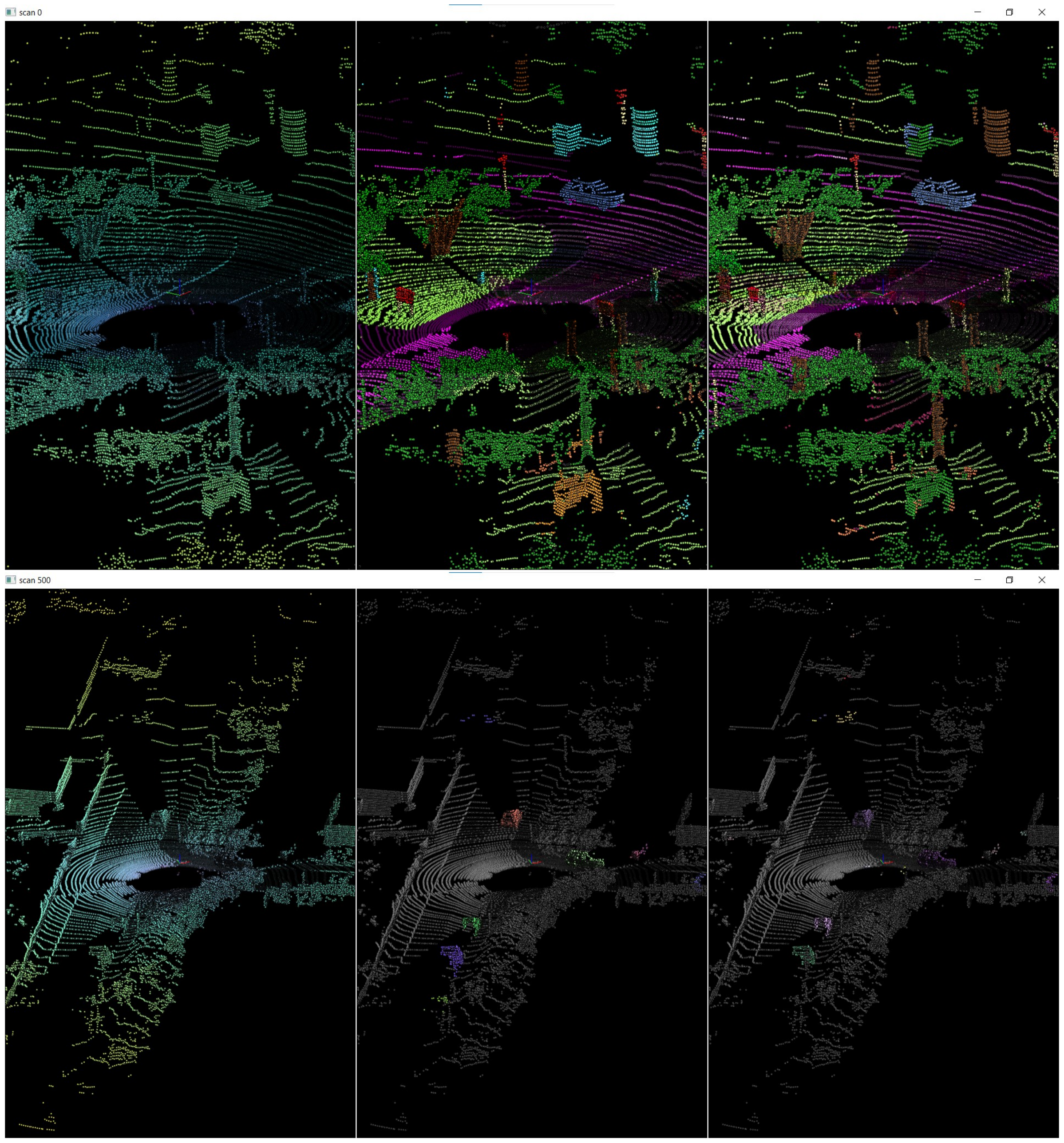

Figure 3. Visualizations of semantic and instance segmentation results. From left to right : LiDAR point cloud, ground-truth segmentation, predicted segmentation

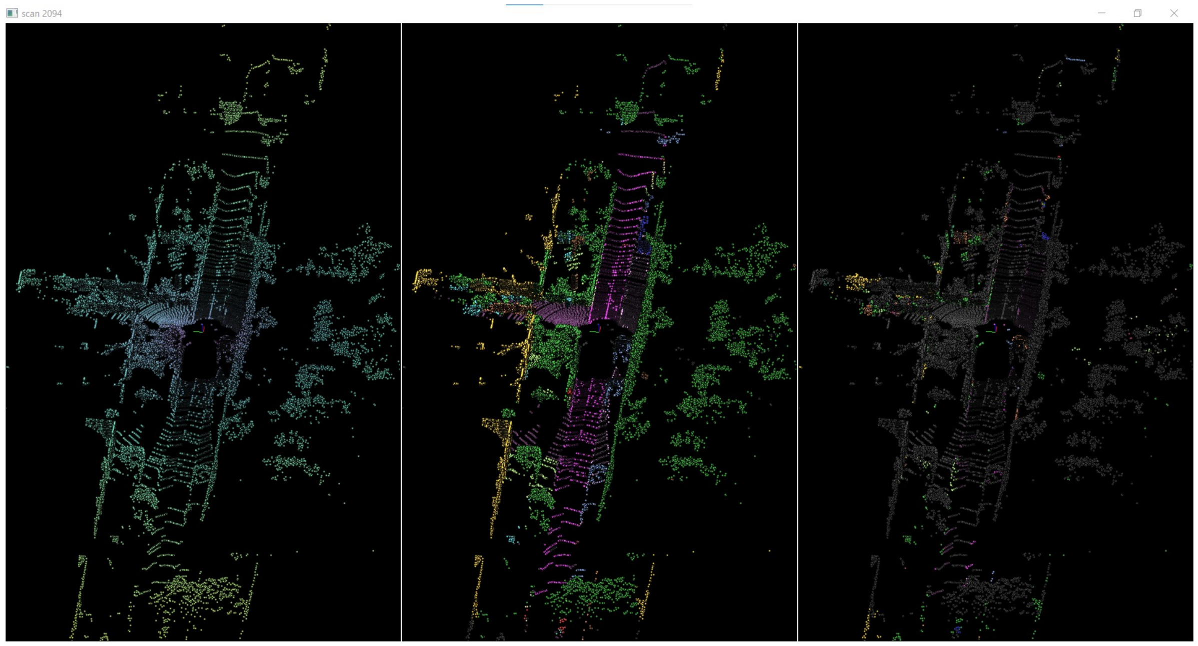

Figure 4. Ground-truth segmentation and difference map (only misclassified points are colorized)

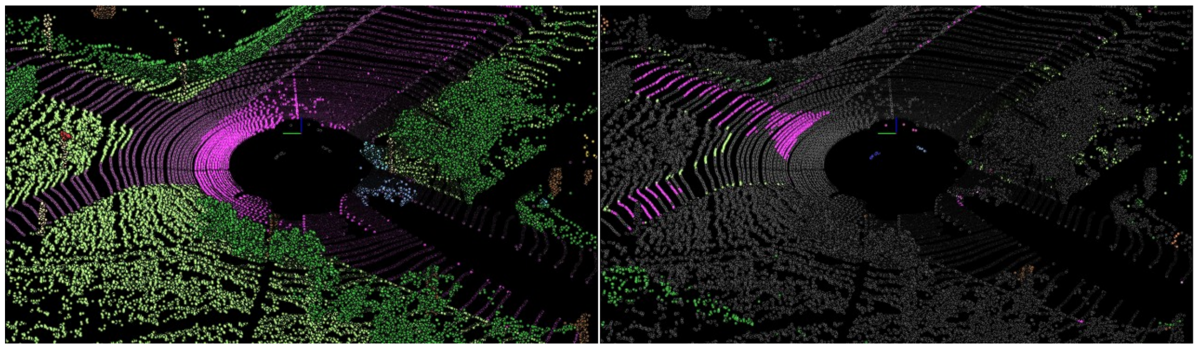

Figure 5. Confusion between sidewalk and road

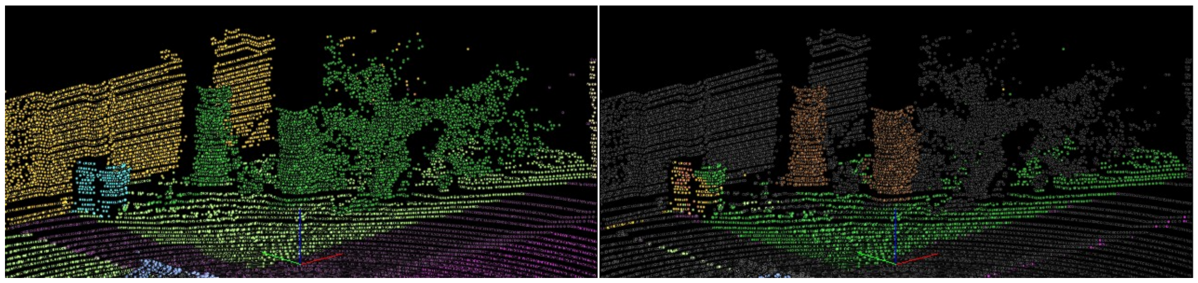

Figure 6. Wrong segmentation of trunks

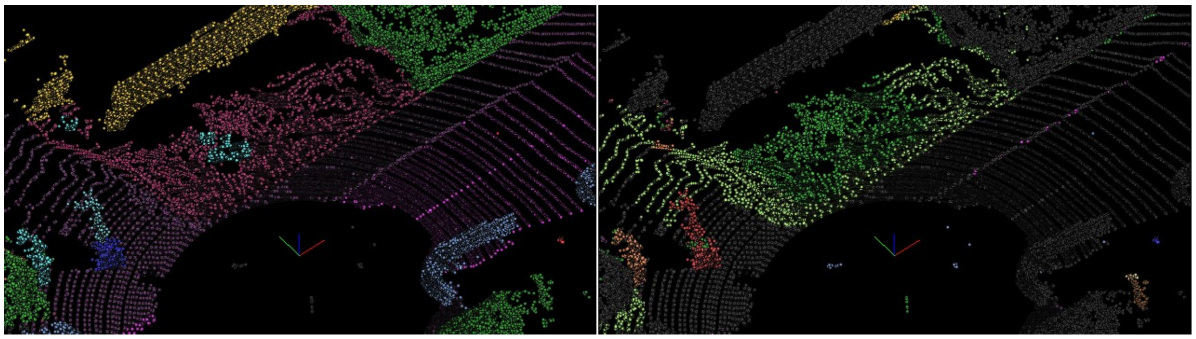

Figure 7. Wrong segmentation of parking area

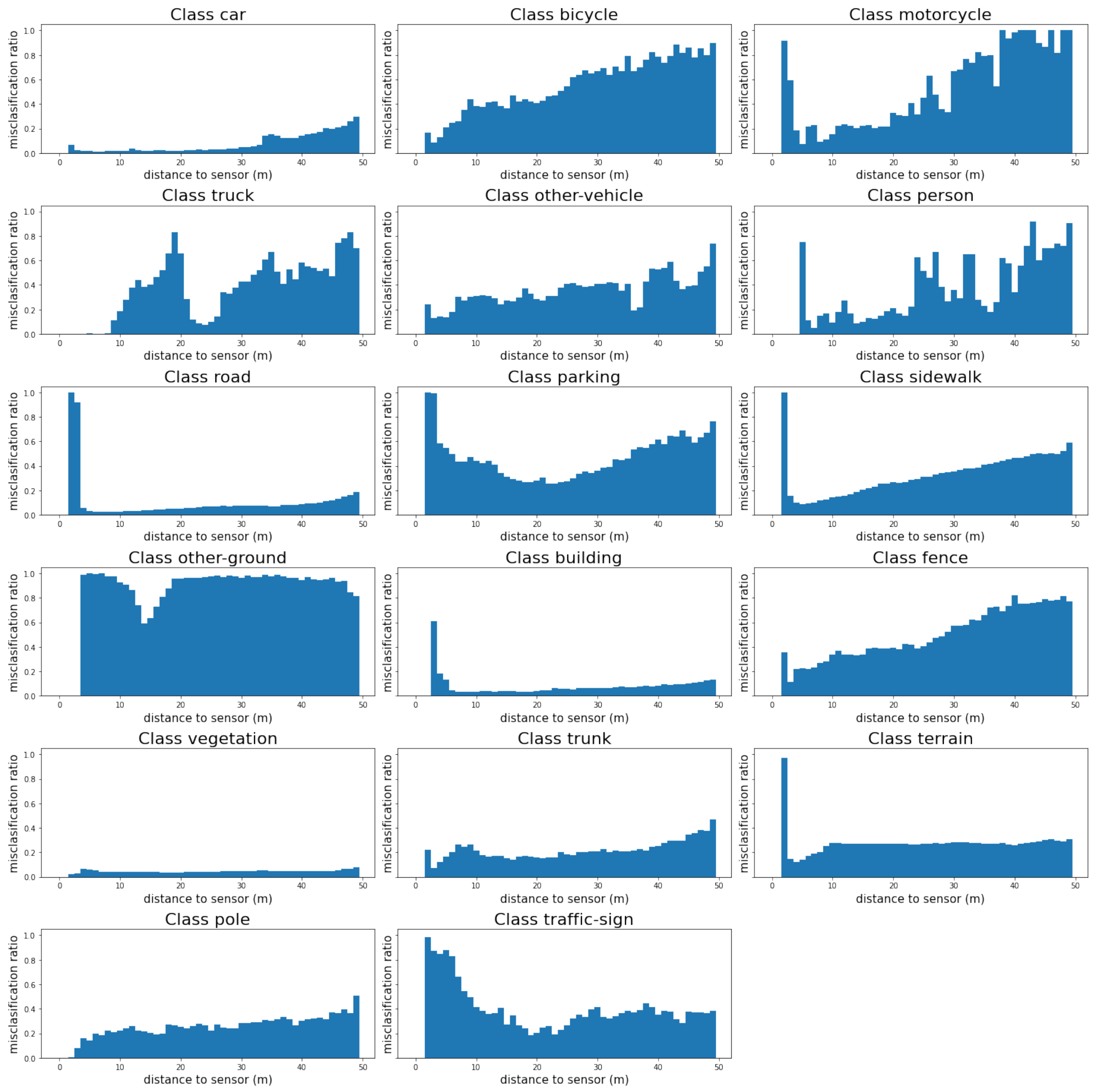

Figure 8. Relative distribution of misclassified points