Reproduction of KPRNet: Improving projection-based LiDAR semantic segmentation

Aden Westmaas, (4825373), a.b.westmaas@student.tudelft.nl
Badr Essabri, (5099412), b.essabri@student.tudelft.nl
Guido Dumont, (5655366), g.dumont@student.tudelft.nl

April 24, 2023

In this blog post, we discuss our efforts to replicate and extend the model and results presented in the paper, KPRNet: Improving projection-based LiDAR semantic segmentation by D. Kochanov, F. Nejadasl, and O. Booij. Our primary objectives are to reproduce the results from the paper, explore the impact of data augmentation on the original SemanticKITTI dataset to test model robustness, and suggest an approach for training the model on KITTI-360. The work showcased in this blog is part of the CS4240 Deep Learning course (2022/2023) at Delft University of Technology.

Contents

1 Introduction

Semantic segmentation is a critical component in the perception systems of autonomous vehicles, as it enables these vehicles to understand and interpret their surroundings. Over the years, significant progress has been made in segmenting camera-based images, and more recently, with the release of publicly labeled LiDAR 3D point cloud datasets, considerable strides have been made in the segmentation of LiDAR measurements as well. LiDAR-based semantic segmentation methods can be broadly classified into two categories: point-wise methods acting directly on the 3D point cloud and methods that rely on the well-developed field of image segmentation. The latter involves projecting individual LiDAR sweeps onto 2D range images, which serve as input to custom CNNs. The resulting 2D predictions are then post-processed using non-learned CRFs or KNN-based voting to obtain more accurate labels for each 3D point.

In this blog post, we explore the KPRNet model, which combines the best of both approaches to enhance the accuracy of segmenting LiDAR scans. The primary contributions of KPRNet are an improved architecture for 2D projected LiDAR sweeps and a learnable module based on KPConv to replace the post-processing step. The blog post shows the steps taken to reproduce the model from the original KPRNet github repository. Furthermore, we enhanced the dataset used for training and testing the KPRNet model by applying data augmentation techniques. This allowed us to assess the model's robustness when presented with this newly augmented data. Lastly, we propose a method to retrain the KPRNet model on the newer KITTI-360 dataset.

Our source code and instructions can be found here.

2 Method

The method proposed in the paper brings together a 2D semantic segmentation network and 3D point-wise layers. The convolutional network receives a LiDAR scan projected as a range image as input. The 2D CNN features obtained are then back-projected to their corresponding 3D points and forwarded to a 3D point-wise module that predicts the ultimate labels. Figure 1 shows the full network architecture. The ResNeXt features with stride 16 are fed into an ASPP module and combined with the outputs of the second and first ResNeXt blocks, which have strides of 8 and 4. The result is passed through a KPConv layer, followed by BatchNorm, ReLu and a final classifier.

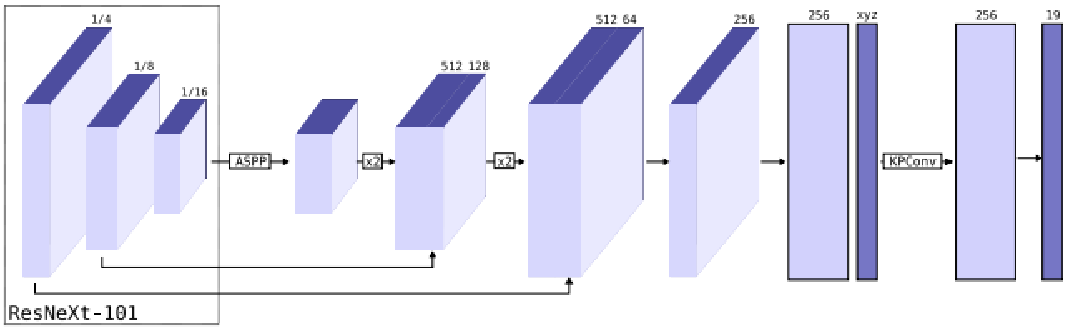

Figure 1: The KPRNet architecture

2.1 2D semantic segmentation

The first part of the model consists of a ResNeXt-101 encoder and decoder. The CNN part of the full network is pre-trained on the Cityscapes dataset as the network must learn semantic features of city like scenery. When transferring the model to the LiDAR segmentation task, we discard one of the filter planes in the first layer because the input for the task only requires two channels, namely, inverse depth and reflectivity.

2.2 3D semantic segmentation

Most methods utilize spherical projection to convert point clouds to range view images. This paper used an alternative method in which the point clouds are unfolded according to the order in which they were captured by the LiDAR sensor, resulting in smoother projections. Despite the remaining discretization artifacts and overly-smooth 2D labels generated by the CNN, back-projecting to the 3D point cloud can result in mispredictions. To address this, RangeNet++ and other 2D segmentation methods employ KNN or CRF post-processing steps. However, finding the right balance between over-smoothing and under-smoothing the 3D labels using these methods can be challenging.

In KPRNet, this post-processing step is replaced by incorporating a single KPConv layer before the final classification. KPConv is a point-convolution operator capable of correcting misclassifications by considering the 2D features of each point and its surrounding 3D context. This change to the CNN architecture is minimal, and the pipeline from a 2D range image to 3D point labels becomes end-to-end learnable.

2.3 Training

The proposed network was trained and evaluated on the SemanticKITTI which contains 21 sequences. Sequences 1-7, 9 and 10 were used for training while sequence 8 was used for validation. Sequences 11-21 were used as the test dataset. The authors trained the network using 8 Tesla V100 GPUs. The dimensions of the input range scans are 64x2048.

2.4 Results of paper

Table 1 shows the test performance of the KPRNet proposed by the paper covered in this blog post. The table shows the test performance for various classes as well as the mean-IOU achieved by the model. From table 1, it can also be seen that for most classes, KPRNet outperformed the state-of-the-art at the time the paper was written.

Approach	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign	**mean-IoU**
SalsaNext [5]	91.9	48.3	38.6	38.9	31.9	60.2	59.0	19.4	91.7	63.7	75.8	29.1	90.2	64.2	81.8	63.6	66.5	54.3	62.1	59.5
KPConv [11]	**96.0**	30.2	42.5	33.4	**44.3**	61.5	61.6	11.8	88.8	61.3	72.7	**31.6**	90.5	64.2	84.8	69.2	69.1	56.4	47.4	58.8
SqueezeSegV3 [13]	92.5	38.7	36.5	29.6	33.0	45.6	46.2	**20.1**	91.7	63.4	74.8	26.4	89.0	59.4	82.0	58.7	65.4	49.6	58.9	55.9
RandLa-Net [6]	94.2	26.0	25.8	**40.1**	38.9	49.2	48.2	7.2	90.7	60.3	73.7	20.4	86.9	56.3	81.4	61.3	66.8	49.2	47.7	53.9
RangeNet++ [9]	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
KPRNet [Ours]	95.5	**54.1**	**47.9**	23.6	42.6	**65.9**	**65.0**	16.5	**93.2**	**73.9**	**80.6**	30.2	**91.7**	**68.4**	**85.7**	**69.8**	**71.2**	**58.7**	**64.1**	**63.1**

Table 1: Quantitative performance evaluation of KPRNet LiDAR semantic segmentation on the SemanticKITTI dataset

3 Reproduction

Our main goal for the reproduction was to reproduce Table 1 from the paper. To achieve this, we cloned the paper's original GitHub repository. Afterwards, we downloaded the required dataset (SemanticKITTI) and the fully trained model weights. From here on, it was challenging to get the code running. The KPRNet repository's README gives insufficient documentation for how to get the code running smoothly. Also, the requirements.txt file, that was included in the repository, did not work as many of the packages were not backward compatible with newer versions of Python. For this reason, we had to create a new Python environment manually and install the packages one by one. Finally, after solving the Python environment, we encountered the next problem, lack of computation resources. A single prediction from the test set using the pre-trained model took over 30 seconds, and each sequence had about 4700 predictions. So, running the test set on our machines locally was not an option, as the test dataset contained 11 sequences. For this reason, we decided to run the code in a Google Cloud virtual machine (VM). This VM was equipped with an 8-core CPU and a NVIDIA P100 GPU. Despite using a powerful VM, running the test dataset on the model still took about 12 hours to complete.

Table 2 shows the IoU scores achieved by the model for all the classes in the test dataset. From this table, it can be seen that our scores are close to what the authors of the paper achieved, but not exacty the same. The authors achieved a mean IoU score of 63.1 while our mean IoU is 61.2. In theory, there should not exist a discrepancy between the achieved IoU scores as we cloned the code repository of the paper and used the exact same test dataset as described in the paper. We discussed this phenomenon with one of the authors of the paper and he suggested that this discrepancy may be caused by using different versions for the libraries.

Network	**Car**	**Bicycle**	**Motorcycle**	**Truck**	**Other-vehicle**	**Person**	**Bicyclist**	**Motorcyclist**	**Road**	**Parking**
KPRNet [Paper]	95.5	54.1	47.9	23.6	42.6	65.9	65.0	16.5	93.2	73.9
KPRNet [Reproduction]	95.2	45.0	50.8	18.8	36.7	65.8	55.7	13.6	93.2	72.1

Network	**Sidewalk**	**Other-ground**	**Building**	**Fence**	**Vegetation**	**Trunk**	**Terrain**	**Pole**	**Traffic-sign**	**mean-IoU**
KPRNet [Paper]	80.6	30.2	91.7	68.4	85.7	69.8	71.2	58.7	64.1	63.1
KPRNet [Reproduction]	79.7	27.8	92.0	68.9	85.7	69.0	70.7	58.3	63.1	61.2

Table 2: Results for the reproduction of Table 1 from the paper

4 Data augmentation

To test the robustness of the model, we employed a data augmentation technique, often called: Pixel Dropout. The model takes a two-channel rangeview as input, containing inverse depth reflectivity. In Figures 2 and 5, you can see the original rangeviews without any augmentation. To augment the images, we randomly set a certain percentage of pixels to a large depth and low intensity, which simulates an incomplete point cloud. As shown in Figures 3-4 and 6-7, we experimented with dropout rates of 10% and 20%, respectively, resulting in noisy images.

The dropout process is accomplished using a random mask that sets the depth and reflectivity to -10. This value was chosen because depth and reflectivity are normalized between -10 and 10, where high depth and low reflectivity

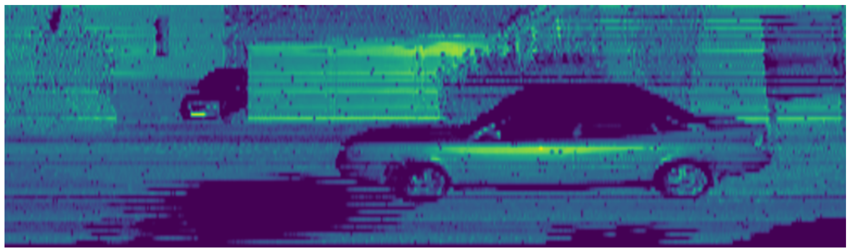

Figure 2: Reflectivity channel of an original range view of a LiDAR point cloud

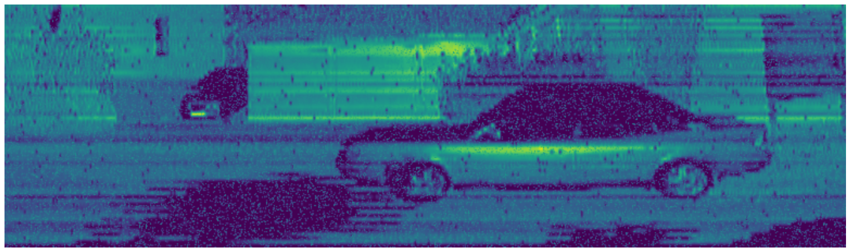

Figure 3: Reflectivity channel of the range view in figure 2, with 10% of all pixel values randomly set to zero

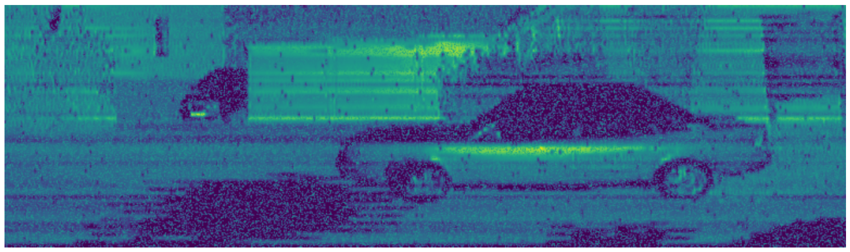

Figure 4: Reflectivity channel of the range view in figure 2, with 20% of all pixel values randomly set to zero

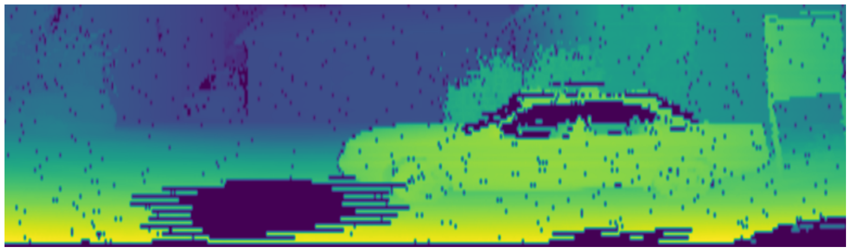

Figure 5: Depth channel of an original range view of a LiDAR point cloud

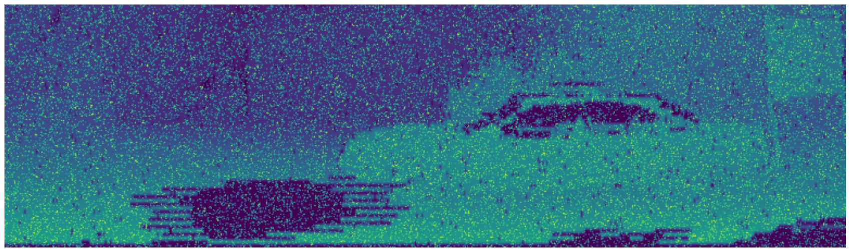

Figure 6: Depth channel of the range view in figure 5, with 10% of all pixel values randomly set to zero

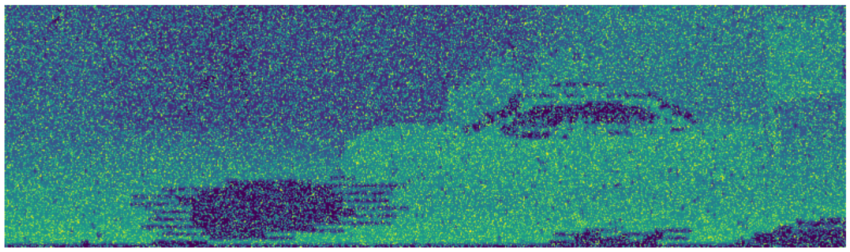

Figure 7: Depth channel of the range view in figure 5, with 20% of all pixel values randomly set to zero

are both represented by -10 (See the pseudocode below). The code for the dropout is modified in the file semantic_KITTI_data_augmentation.py in our GitHub repository. By adjusting the "obscure_factor" , the dropout percentage can be controlled.

```
# Determine the total number of pixels in the image
height , width = depth_image.shape
num_pixels = height * width

# Set the desired percentage of pixels to 0
percent_zero = 0.1
num_zeros = int(num_pixels * percent_zero)

# Generate a mask array to indicate which pixels should be set to 0
mask = np.zeros(num_pixels , dtype=np.uint8)
mask[:num_zeros] = 1
np.random.shuffle(mask)

# Reshape the mask array to match the image dimensions
mask = mask.reshape((height , width))

# Set the corresponding pixels in the image to -10
depth_image[mask == 1] = -10
refl_image[mask == 1] = -10
```

We evaluated the performance of the model with and without data augmentation. The results show a significant decline in performance across all classes when data augmentation is used, as seen in the score tables below. Smaller objects, such as people and motorcycles, were particularly impacted. We believe that the decline in performance is due to the depth image, where object shapes become less discernible even at a relatively low dropout rate. This suggests that the depth information is critical to the model, relative to the reflectivity, as the shapes of objects can still be seen in the reflectivity images.

Network	Car	Bicycle	Motorcycle	Truck	Other-vehicle	Person	Bicyclist	Motorcyclist	Road	Parking
KPRNet [Paper]	95.5	54.1	47.9	23.6	42.6	65.9	65.0	16.5	93.2	73.9
KPRNet [10% augmented data]	26.7	0.0	0.0	0.0	0.9	0.6	0.2	0.0	2.3	0.0
KPRNet [20% augmented data]	13.4	0.0	0.0	0.0	0.3	0.1	0.0	0.0	1.0	0.0

Network	Sidewalk	Other-ground	Building	Fence	Vegetation	Trunk	Terrain	Pole	Traffic-sign	mean-IoU
KPRNet [Paper]	80.6	30.2	91.7	68.4	85.7	69.8	71.2	58.7	64.1	63.1
KPRNet [10% augmented data]	4.7	0.0	32.2	5.9	49.4	9.3	11.2	13.6	37.4	10.2
KPRNet [20% augmented data]	7.1	0.0	25.1	5.6	41.1	5.8	6.7	8.8	31.0	7.7

Table 3: Results for KPRNet tested on the augmented dataset

5 Implementation of the KITTI360 dataset

The current model is trained and validated using the SemanticKITTI dataset. However, soon after the paper was published, a new dataset called KITTI360 was released. This dataset is more extensive compared to SementicKITTI and contains 73km of labeled camera and LiDAR data (recorded with the same LiDAR sensor). In this section, we present a methodology to train and validate the model on the KITTI360 dataset. Beause only a hand full of datasets contain segmentation on point-level bases, making the KITTI360 dataset compatible with the model will increase the size of the trainingset significantly and thus potentially improve performance as well.

5.1 SemanticKITTI vs KITTI360 dataset

The main difference between the two datasets is the type of LiDAR data available. SemanticKITTI consists of single point clouds created by a single revolution/sweep of the sensor, including ego-motion compensation. KITTI360 however, consists of sequences where multiple point clouds are merged together to construct an entire sequence. From these sequences, it is not possible to directly construct a rangeview, which is required as input for the KPRNet. So, to use KITTI360 within KPRNet, a data preparation step is needed to construct rangeviews of lidarpoints obtained by a single revolution of the LiDAR sensor.

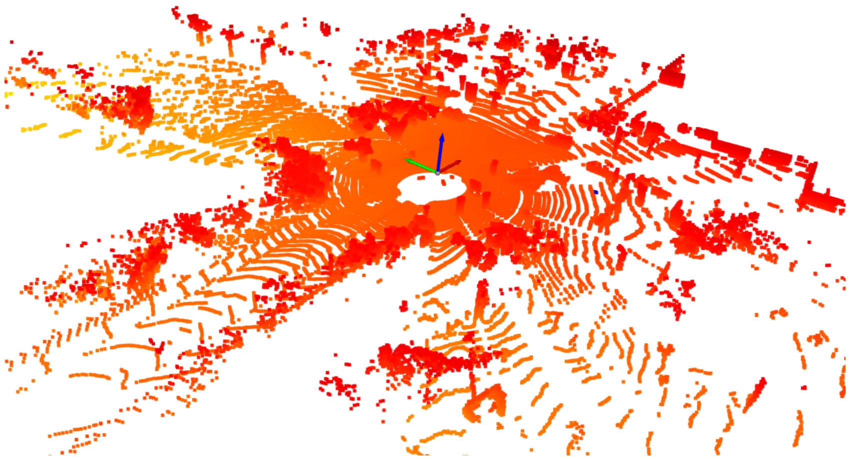

Figure 8: Pointcloud in SemanticKITTI

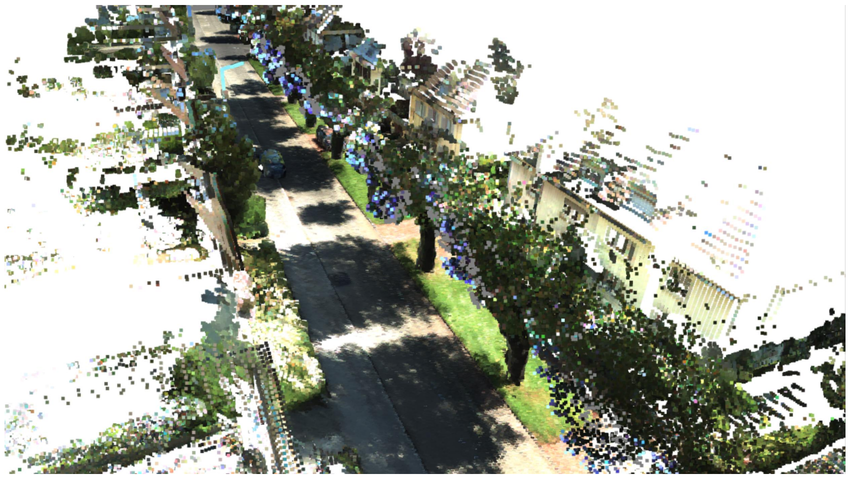

Figure 9: Sequence of connected pointclouds within KITTI360

5.2 Pointcloud sampling

To create single point clouds from the sequences in the KITTI360 dataset, a sampling method was created. This sampling method samples points from a sequence of LiDAR pointclouds based on its origin (position of the LiDAR sensor) and the characteristics of the sensor, see the pseudocode below. One iteration creates a single LiDAR pointcloud that *could* be measured by the LiDAR sensor if it was positioned in the same place as the used origin. The origin used in this sampling method is calculated from the car pose data provided in the KITTI360 dataset.

```
pcl = KITTI360 # KITTI360 Lidar pointcloud
pcl_transformed = transform(pcl, origin) # Transform pointcloud with respect to origin
pcl_spherical = spherical_coordinates(pcl_transformed) # Convert to spherical coordinates (r, phi,
    theta)
pcl_spherical = pcl_spherical[:, 0] < max_range_lidar # Sampled points within range of the sensor

# Define horizontal and vertical angles
```

```
horizontal_angles = np.linspace(lidar_horizontal_range , lidar_horizontal_resolution)
vertical_angles = np.linspace(lidar_vertical_range , lidar_vertical_resolution)

# Sample instances
for h in horizontal_angles:
    for v in vertical_angles:
        point_index = np.argmin(abs(pcl_spherical[:, 1:] - [v, h]))
        cloud_indeces.append(pcl_spherical[point_index])
```

The pseudocode above creates ‘perfect’ point clouds where every light beam sent receives a measurement. In practice, however, many emitted lightbeams do not reflect back to the lidar sensor, resulting in a suboptimal point cloud. This phenomenon is simulated in the sampling method by randomly deleting a portion of the lidar points.

5.3 Results and limitations

This sampling method creates point clouds that could be measured by the lidar sensor, see the figure below.

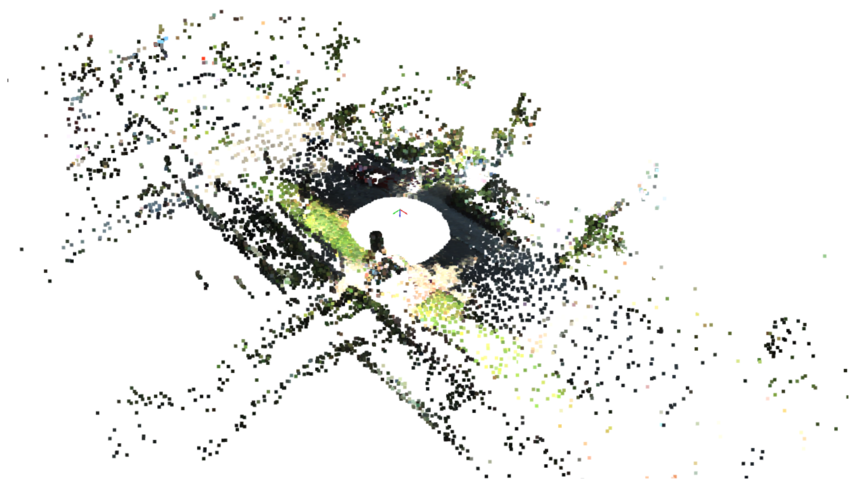

Figure 10: Sequence of connected pointclouds within KITTI360

However, the sampling methodology created is not perfect. As mentioned before, the resulting point cloud could be measured by the LiDAR sensor and therefore contain inaccuracies. Further, the current version of the sampling methodology is computationally expensive, so it’s not a plug-and-play algorithm to make the entire KITTI360 usable for the KPRNet. This is also the reason why the model is not evaluated on these sampled point clouds.

6 Conclusion

In conclusion, this blog post discussed our efforts to reproduce and extend the KPRNet model proposed in KPRNet: Improving projection-based LiDAR semantic segmentation, for projection-based LiDAR semantic segmentation. We reproduced the results from the paper and explored the impact of data augmentation techniques on the SemanticKITTI dataset to test model robustness. We found that noise in the inverse depth channel of the rangeview has a greater impact on the model’s performance compared to similar noise in the reflectivity channel. Additionally, we proposed a method to use the KPRNet model on the newer KITTI-360 dataset. However, the proposed sampling methodology is computationally expensive and thus not a plug-and-play solution.

6.1 Limitations

We did not manage to complete a full reproduction of the paper as we were unable to train the model from scratch. As mentioned before, the authors of the paper used many computational resources to train the model on the semanticKITTI dataset. We did not have this amount of compute at our disposal, and thus, it was impossible to train the model on our laptops locally or on the Google Cloud Platform. Furthermore, the proposed sampling methodology to use the KITTI360 dataset has limitations. Firstly, it constructs a LiDAR sweep that could be measured by the LiDAR sensor. Secondly, the resulting point clouds can contain imperfections due to the assumptions made in the sampling process.

7 Contribution

Aden: Reproduction
Badr: Data augmentation
Guido: Implementation of KITTI360 dataset