Investigation of Machine Learning Techniques in Detection of Cones in LiDAR Data for an Autonomous Race Car

Oliver Sanchez, Bc. student NTNU

I. Introduction

A. Background

In this investigation, multiple machine learning techniques will be initially explored and subsequently evaluated in their ability to predict cones from noisy LiDAR data. The data was collected during an autonomous run of a race car developed by the formula student team Revolve, where the author works as an autonomous systems engineer. As this is an exploratory investigation, the overall goal is not to achieve any particular end result but rather to complete a comprehensive study of the feasibility of improving a detection pipeline, which includes nearest neighbor clustering and simple cone dimension filtering with a more complex solution. When this investigation commenced, no specific models were chosen in order to keep the investigation exploratory and allow for agile decision-making throughout. The approach to model selection focused on the least complex models initially, gradually increasing the complexity of the model—a kind of model-based curriculum learning. At the conclusion of the investigation, the models chosen were: Support Vector Machine, Fully-connected Neural Network, U-Net, and Graph Neural Network.[1]

B. Motivation

Currently, the autonomous race car has a well-performing detection pipeline. It is built to downsample and segment the ground plane with RANSAC before it employs simple filtration to choose which clusters are actually cones. The detection pipeline performs well, and the race car has executed multiple high-speed runs using it without issue. However, pursuing new solutions and pushing performance is what racing is all about. This spirit underpinned this investigation, which aims to develop a machine learning model that could, within a few milliseconds, provide exact coordinates of the cones given only a downsampled and groundplane segmented cloud. Such a model could significantly decrease runtime, making it plausible to integrate other more computationally heavy solutions. Furthermore, this is somewhat a big data solution—the more data gathered from the car, the more the model should learn and the better it should drive in the next run, albeit this is only in principle and assumes a scalable architecture. The current detection pipeline does not learn and does not improve over time, which is a reason why it can be deemed an inferior solution.

C. Related Work

The majority of work using SVM and U-NET architectures with LiDAR data focuses on aerial data, which can be treated as 2D if one does not consider the intensity, but that is not very relevant to this investigation. This investigation is not sufficiently sophisticated to warrant a deep dive into all related papers. However, two papers are worthy of mention. One paper introduces an Auto-Registration technique that learns how to standardize each node (point) as to be invariant to the positioning or tilt of the LiDAR (Weijing & Ragunathan, 2020 [2]). This invariance can become a significant advantage as the graph neural network they propose, Point-GNN, can focus on learning the geometric configurations of the data and therefore be a better model. This is similar to how the geometric relationships between neighboring points are encoded as features in this investigation's GNN. It is worth noting that the Weijing & Ragunathan paper was focused on 3D classification, while this investigation is dealing with 2D segmentation; nonetheless, the techniques

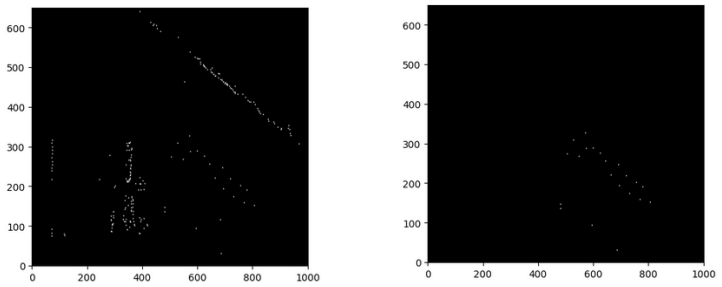
[1]The initial proposal for this project heralded a pointnet type of architecture as the holy grail of this investigation. This architecture was not explored due to time constraints with labeling the data, as 3D data labeling is a lot more time-intensive than 2D labeling.

and architectures that work for one task often work well for the other. The other relevant paper proposed a LiDAR U-Net model for semantic segmentation, which is exactly the task this investigation details (Biasutti et al, 2019 [1]). One of the key ideas in the paper is embedding features from the initial 3D data into the 2D project, an idea that was not incorporated in this investigation but will be encouraged as potential future work to improve the model even further.

II. Methods and Results

A. Data Collection

Gathering the data had a simple first step: Drive the race car autonomously and record a ROS bag filled with point cloud data. The second step, however, required more rumination. Looking into the best way of annotating point cloud data and considering the amount of code and uncertainty in developing an automatic labeling pipeline that would use the SLAM approved cones as ground truth, it was decided more reasonable to focus on 2D data. Once it was clear this investigation would focus on 2D projected data the choice had to be made of where in the pipeline to introduce the model. The earlier it is introduced the more it could revamp the runtime and detection accuracy, but it would also require much more data and a more complex model to run. Considering that the downsampling of the point cloud requires little processing power and only reduces the points to voxels – there is no real data lost, just perhaps accuracy in estimating the true center of the cone. Additionally, considering the groundplane segmentation of the already downsampled cloud – there is a lot of noise reduction to be gained and a relatively low runtime. Therefore, it was deemed most reasonable to place the model after the LiDAR point cloud has been downsampled and a groundplane segmented, and then finally projected to a 2D that is horizontal to the LiDAR. This leaves a rather clean data, roughly 2,000 points with a lot of outliers and varying amounts of noise that the groundplane segmentation did not catch (see Figure 1 a) for an example of the data at this stage). Due to the time consuming nature of the labeling process, in conjunction with the author only having two hands and one computer, only a limited amount of data was annotated – unfortunately all of the data was from the same run as the annotation pipeline was tailored to it, this is a big limitation on the generalizability of the results[2].



(a) After downsampling/segmentation (b) Labeled data

Fig. 1: Results with SVM

B. Support Vector Machine

The idea behind this model was simple, support vector machines are in essence a multidimensional classifier and we have a geometry based problem. Each point was encoded geometrically by using its distance to relative neighbors, initially only 5 but went as high as 20 neighboring distances. Geometrically each cone should have one neighbor of an almost consistent distance, the cone opposite it on the track. Then each cone has one cone on each of its sides, them being roughly the same length, and then finally there are roughly two and two cones at approximately equal distance from the cone in question. This, however, works perfectly if the track is straight and all the cones are still in question, which is not the case in the vast majority of cases. With more realistic inputs the idea still holds, the track does have a nice geometric relation between the cones, but not the noise, hence a support vector machine should theoretically be able to yield some reasonable accuracy in prediction. To visualize this geometrical pattern that is present in the track layout, the sum of the first five distances was plotted for each point (see Figure 2).

[2]See Figure 1 for examples of the raw data and the labeled data. In total 24 samples were annotated. See github for the entire collection of these.

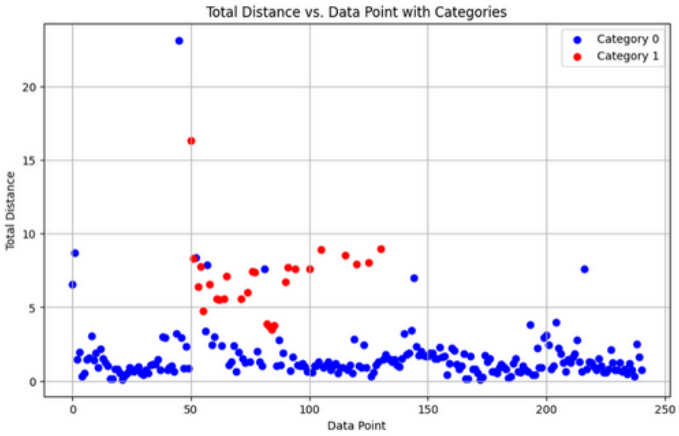
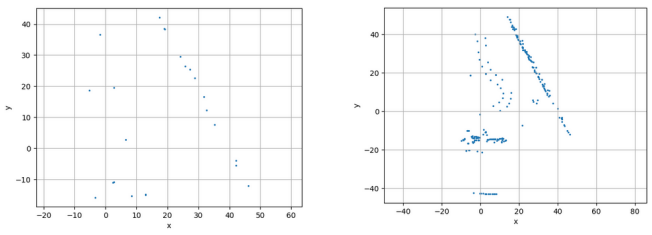


Fig. 2: Scatter plot of the total distance of five shortest distances for each cone

Despite the indication by the preprocessing that the SVM could produce great accuracy, the results were lackluster. The model failed to capture the geometrical relationships and seemed to only really notice the straight line patterns (see Figure 3 for a plot of the input and prediction of the model). The SVM only managed a poor 89.06% on a 20% test set, that is, it managed to classify correctly 89.05% of the roughly 300 points but only 3 of those correct classifications were true positives, there were 22 false negatives. This was considered strange as the data it was trained on specifically did not have a plethora of straights, and conversely had training data with noise patches that were straight in nature labeled as non-cones. The conclusion on this model was simple. The geometric relationship of the cones, considering the curve-featured track and noise both on the track and in the background, the model was too simplistic (see github repository for the entire preprocessing and code).



(a) Inference with SVM (b) Input data

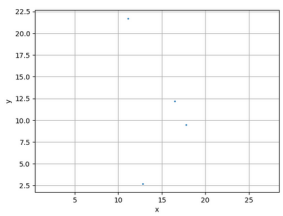
Fig. 3: Results with SVM

C. Fully-Connected Neural Network

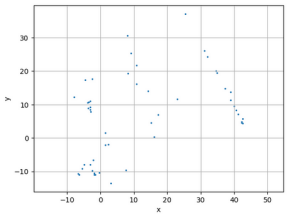
The classical problem of using neural networks on LiDAR data is that the size is not consistent between different point clouds. One point cloud might have 2,000 points and the next 2,500 and even if the fluctuations were less, a fully-connected neural network has a standardized size of inputs, making 2,001 inputs also problematic for a network trained on 2,000 inputs. The reason for this inconsistent LiDAR output is innate in the technology as depending on distance and complexity of objects some laser outputs will not return to the LiDAR and get detected as a point in the environment. The employed network therefore standardized the amount of points to 600, simply filling the missing points with 0 values. This simple fix is no solution, as the network struggles to establish weights to nodes as it cannot know if an input is a 0 or a coordinate. The data was flattened as to input vectors of the following form: input = [x1, y1, x2, y2, ... , 0, 0], len(input) = 620. The network was a classically simple fully-connected neural network using BCE loss, ReLU activations, and an Adam optimizer. The simplicity of the model was devised with the plan in mind of optimizing it further if initial results motivated behavior as such. The results did not motivate anything of the kind, the model had some very slight decrease in loss. In 10 epochs it went down from 0.75 to 0.69. The BCE resulted in a very conservative model, while using cross-entropy (CE) gave slightly more positive results. The CE results, however better, were not enough to warrant further work on this architecture (see Figure 4 for visualization of the performance of each model).

D. U-Net

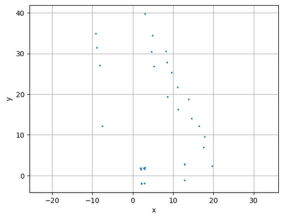
The data being projected onto a 2D plane makes it very reminiscent of an image, and when working with images convolutional neural networks are always an obvious choice. Seeing as we are performing semantic segmentation by classifying individual pixels the U-net architecture was assessed as a perfect fit for the task. The possible issue with the U-net is its complexity, it is a network the author was not exceedingly familiar with at the outset of the investigation. The complexity, furthermore, introduces some runtime concerns, considering the inference should work online (when the race car is driving) and in ideally less than 10 ms with



(a) Inference w/CE Criterion



(b) Inference w/BCE Criterion



(c) Label

Fig. 4: Results with Fully-Connected NN

the current detection pipeline. The mix of shallow layers that excel in capturing local patterns and spatial correlations, and deep layers that are able to understand more complex and larger correlations. The detailed shallow layers have relatively small kernels that can only detect the local patterns of the points, such as being isolated and possibly seeing a few other points in the relatively far distance, as opposed to clusters of noise. The shallow kernels are applied to more detailed representations of the data and hence are finer, while the deep layers pool at lower levels making the layers more coarse and not manipulating each pixel value directly but rather an aggregated and manipulated representation of the initial pixels. If you imagine the U-Net as its namesake U shape you can think of the deep layers as the bottom of the U shape. This makes the kernels of the deep layers much larger in the degree of original pixels that they cover. Therefore, these deep layers ideally give the model the ability to sense the overall track and how it is shaped, connecting all the small local features into a larger structure that it understands (in some sense of the world). Conceptually the model seems fit for the purpose and considering the impressive results U-Nets have had in medical imaging it is the author's belief that it is only a question of implementing the architecture sufficiently well, i.e., the network complexity should not be a great bottleneck. Initial results with a straightforward U-Net resulted in some results that improved on previous models but nothing close to satisfactory. The model's loss dropped quite significantly but did not seem to quite get the track understanding that was conceptually imagined (see figure 5 for visuals of the early results). At this point it was thought that the model was essentially too simple to grasp the structure, so more layers were added in addition to residual connections (code can be found in github repository and older iterations will be by request). The performance increased further giving very promising results, and now it was just a question of tweaking the model, increasing further would be unwise as it was understanding the structure (and google colab gives a maximum of 15 GB VRAM for free, as can be seen in Figure 5:( ) (see Figure 6 for results).

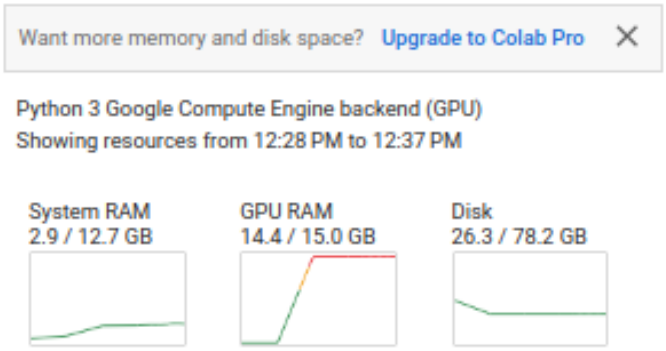
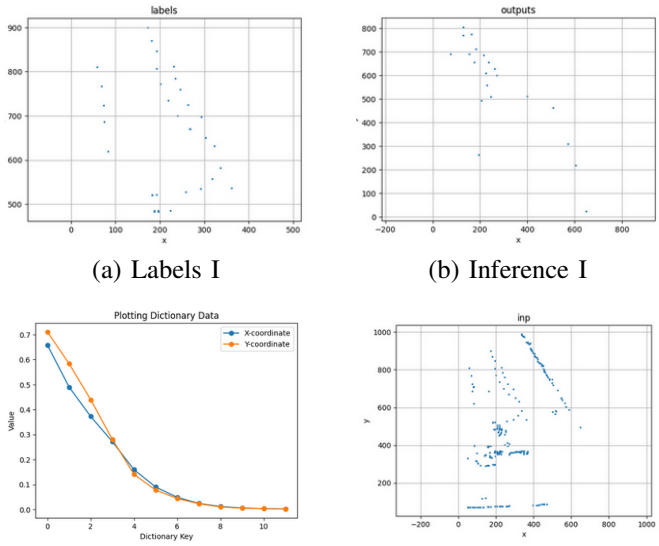


Fig. 5: Figure 7. showing the unfortunate consequences of doing research on a budget

The final mode used dilated convolutions in order to get an even larger kernel for the deep coarse layers of the model. This was an informed decision based on viewing the performance of the previous architecture, the idea was for the model to comprehend the entire track and this way not only output parts of it. The results for this model proved to be a significant improvement over the previous model (with still virtually the same size!). Finally having proper long-dependence understanding and completing the track in the entire validation set and only including some noise on the outside of the track (see Figure 7 for results). The noise was mostly reasonable as it resembled the geometrical relationship of a track, and therefore the model predicted it to be a track. To further establish the feasibility of implementing this solution online when the car is driving the inference time was timed and on average with a T4 GPU (the GPU on the car will certainly be less powerful) the inference takes



(a) Labels I (b) Inference I

(c) Training & Validation Loss (d) Input I

Fig. 6: Results on improved model

4.9 ms.

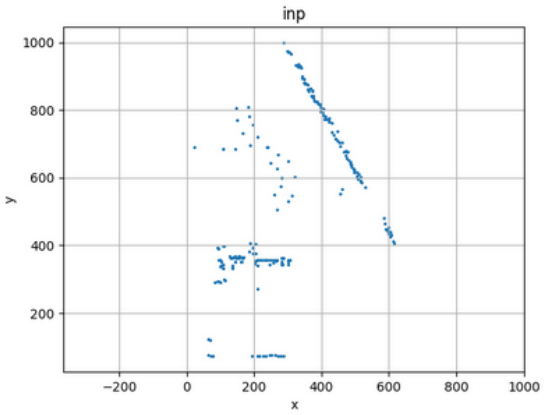
E. Graph Neural Network

In order to encode geometrical features and having input size be invariant in the model, a graph neural network seemed reasonable. A GNN model was created using the coordinates of each input as the nodes, the adjacency matrix had encoded the five nearest neighbors with length of edge. The idea being that each node would get the coordinates of neighbors weighted by the distance between them, and the network had a decent loss from 2.1 to 0.3 in 10 epochs, but much like the fully-connected neural network it showed no understanding of the geometry and seemed to output predictions seemingly randomly and the training was simply the model reducing the amount of predictions it made as there were such few true labels in the dataset. More advanced networks could be devised but it does not seem feasible that any future implementation would perform better than the U-Net performance detailed earlier.

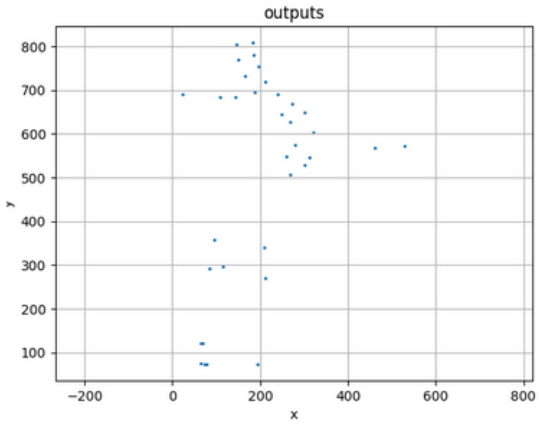
III. Conclusion

This comprehensive exploratory investigation into employing machine learning techniques for predicting cones from LiDAR data for an autonomous race car, has resulted in some insightful conclusions. Concluding on all the models. Firstly, the SVM, while theoretically promising due to the geometric nature of the problem, it proved insufficient and was only able to capture a mere fraction of the necessary relational intricacies within the data. This shortcoming underscored the model's inadequacy in handling curve-featured tracks and diverse noise, leading to its abandonment in favor of more nuanced architectures. Secondly, the fully-connected neural network, similarly exhibited limited potential. Despite data standardization helping with the variable input from the LiDAR data, the inability to distinguish zero-filled data points from meaningful coordinates rendered the model conservative and inconsistent. The model also seemed to lack the complexity to connect the different local features of the track and hence the architecture was inadequate for further pursuit. Thirdly, the graph neural network did not seem to be able to learn much of anything, but the author suspects the architecture was at fault. It is the conclusion that graph neural networks could be a point of research in the future of the detection pipeline improvement it is unlikely to match the performance of a U-Net and should therefore not be prioritized without good reason. Lastly, the more interesting results came using a U-Net architecture. Through a blend of shallow and deep layers, the network began to discern local and global patterns within the point clouds. The incorporation of dilated convolutions brought us closer to a model with practical implications, demonstrating the model's potential in comprehending the complete structure of the racetrack. These results lay the groundwork of the feasibility of reimagining the detection pipeline that allows Revolve's race car to perform autonomously to a high level. By simply having more extensive data the model can both improve in accuracy and in generalizability. Extensive testing across different tracks is also imperative in evaluation of the model and will be fulfilled by annotating more data. The inference time of 4.8 ms is a significant proof of concept as the model is much faster than the current pipeline and could possibly yield better results. Furthermore, the model will simply get better with more data (to some degree) and hence its use compounds with time. This iterative process of model refinement, underscored by the agile approach to model selection, has not only yielded a promising direction with the U-Net architecture but also emphasized the importance of continued exploration in machine learning applica-

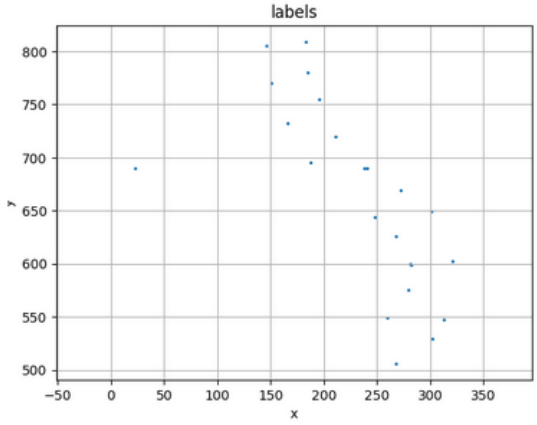
tions. While the final iteration of the U-Net model showcased an impressive milestone, achieving long-range understanding and robust track predictions with robustness to noise, it has also set the stage for future research. In light of the promising results, it becomes evident that a continued investment in data labeling, possibly also 3D, and further architectural optimizations could pave the way for a ML-based detection pipeline that is not only more accurate but capable of learning and evolving over time. This holds the potential to significantly decrease runtime, which could open up to the possibility of integrating more computationally demanding strategies around it, embodying the very ethos of racing—incessant improvement and the relentless pursuit of speed.



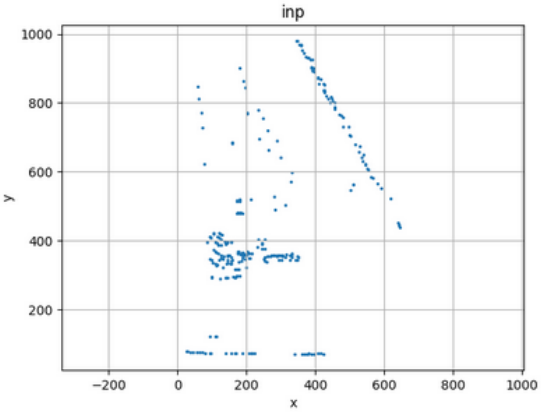
(a) Input I



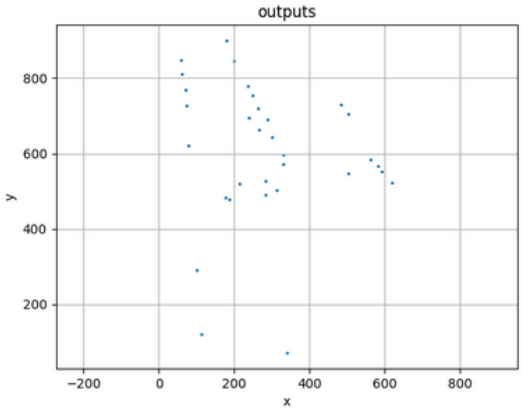
(b) Inference I



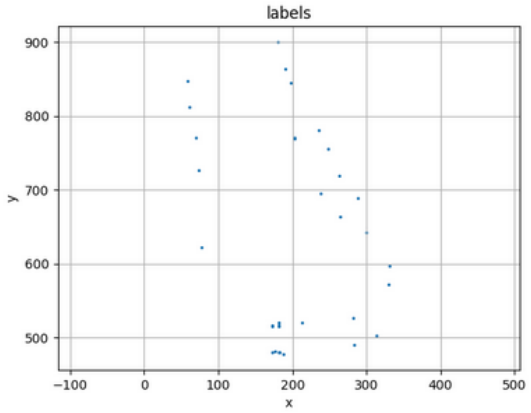
(c) Labels I



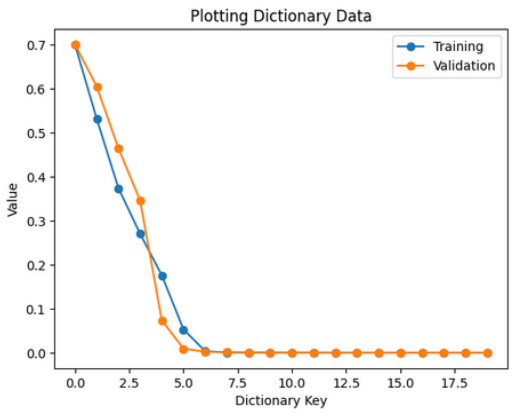
(d) Input II



(e) Inference II



(f) Labels II



(g) Training and Validation loss through 20 epochs

Fig. 7: Results on best model

ACKNOWLEDGMENT

The author would like to thank the Revolve organization for providing not only the data for this investigation but the opportunity to learn and grow there as an autonomous systems engineer. A special thanks to all the previous members of autonomous systems who initially created the excellent detection pipeline that is currently contributing to the race car's autonomous performances.

REFERENCES

[1] P. Biasutti, V. Lepetit, J.-F. Aujol, M. Bredif, and A. Bugeau, "Lu-Net: An efficient network for 3D Lidar Point Cloud Semantic segmentation based on end-to-end-learned 3D features and U-Net," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 1–10. [Online]. Available: https://doi.org/10.1109/iccvw.2019.00123

[2] W. Shi and R. Rajkumar, "Point-GNN: Graph neural network for 3D object detection in a point cloud," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 1–10. [Online]. Available: https://doi.org/10.1109/cvpr42600.2020.00178

IV. APPENDIX

The entire code used in this investigation is available in the following GitHub repository: ML Detection Pipeline Repository