CS230

Analysis of the Effectiveness of Temporal Point Cloud Data for Object Classification and Perception in Autonomous Vehicles

Matei Armanasu
Stanford University
armanasu@stanford.edu

Shrividya Manmohan
Stanford University
shrivid@stanford.edu

Yuhao He
Stanford University
yuhaohe@stanford.edu

Abstract

Most algorithms for classification and semantic segmentation of 3D point clouds ignore readily available temporal information, motivating us to explore the effects of this information in object detection. We evaluate extending standard architectures to include timestamp data, creating a $4 + C$ dimensional data input to a network when C other features are included. We show that for pre-existing architectures, treating time as an extra feature outperforms simply adding more data, and discuss the difficulties of creating and evaluating a recurrent network with the current state-of-the-art 3D encoders.

1 Introduction

3D object detection is a necessary component of autonomous driving systems. Stories of crashes and fatalities with autonomous vehicles often describe classifications "flickering" or quickly changing between different object types, resulting in the vehicle being unable to make a correctly informed decision about how to proceed. Comparatively, drivers use a short period of time to observe (and react), and do not need to re-predict the existence of nearby vehicles. This idea of continuous observations inspires us to explore if temporal/sequential information of 3D objects can improve the performance of object detection. While object detection models in OpenPCDet use point clouds generated from a union of randomly sampled sweeps, our model uses a sequence of point clouds that are sampled consecutively in time. Regardless of different approaches, both types of models use point clouds as inputs and output bounding boxes, class labels, and objectness scores. We use PointPillar[3] as our baseline 3D detection model, and use a recurrent spatial encoder network in parallel to generate temporal features.

2 Related work

3D Point Cloud Detection

3D object detection has been a critical area for autonomous driving. Datasets like kitti [1] provide point cloud snapshots of traffic scenarios collected by a laser scanner and many models such as Part-A^2, SECOND, PointNet++ and PointPillar utilize these datasets [2, 3, 4, 5, 6]. Despite their differences in approaching object detection, traditional models put emphasis on detections within individual point clouds and few of them incorporate temporal information.

CS230: Deep Learning, Fall 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Temporal Point Clouds

In recent years, datasets with explicitly temporal LIDAR sequences such as nuScenes lidarseg were released [7]. There were some efforts to adapt previous 3D detection models on these datasets; however, instead of pursuing the spatiotemporal relations, these approaches take timestamps merely as an extra feature in addition to x, y, z coordinates. On the other hand, some studies try to explore the benefits of using multiple surrounding frames of inputs [8, 9]. For instance, TPointNet++ aggregates information over time to construct object shapes and build generalizable continuous representations/encodings of these objects, so that it can perform detections by sampling from the sequence of these encodings given spatiotemporal information of inputs [8]. As another exploration of leveraging temporal information, our study evaluates whether directly encoding of sequences of point clouds can improve the performance of object detection.

3 Dataset and Features

We are using the nuScenes dataset, which contains a series of LIDAR sweeps from self-driving car sensors stored as 3D point clouds. The full nuScenes dataset contains 700 scenes for training and 150 scenes for validation. We are primarily interested in the LIDAR and keyframe data from nuScenes. Each scene contains 20 seconds of LIDAR sweeps at the frequency of 20HZ and in total nuScenes contains 1.1 billion annotated points across 34,000 point clouds. Each point cloud consists of a number of successive sweeps, with approximately 26,414 points per sweep (with some variance due to imprecision when removing points that fall on the vehicle itself.)

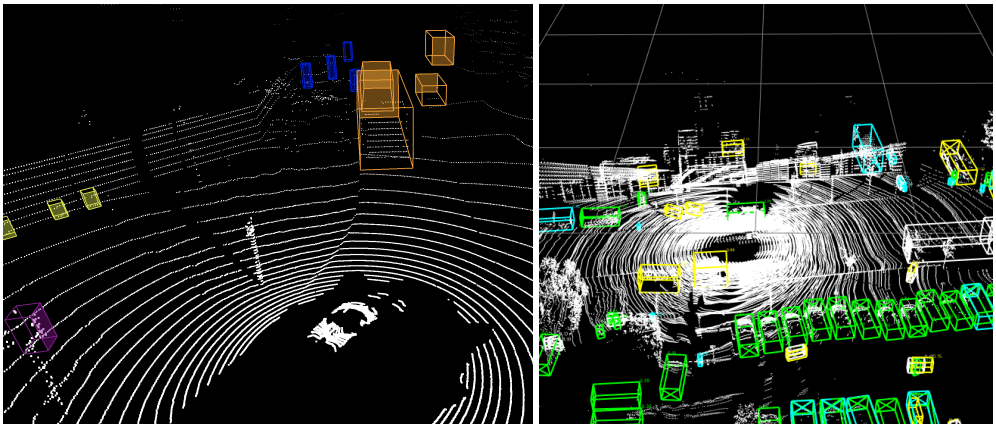


Figure 1: Left: visualization of a single sweep from nuScenes. The colored boxes are ground truth boxes that mark the segmentations. Right: visualization of a full point cloud input created by OpenPCDet through concatenating 10 sweeps, and undoing the egocentric motion of vehicles.

We process nuScenes data via OpenPCDet's data handling pipeline [10]. The result is a 3D point cloud constructed from successive sweeps over a span of time, with each point retaining two feature dimensions representing the intensity of the LIDAR response as well as the time lag relative to the ground-truth annotated sweep (i.e. each point is represented as <x, y, z, intensity, time delay>). OpenPCDet constructs the data in this way in order to develop a denser point cloud representation to feed into different models without explicitly exploring the effect of including temporally-separated data. We explore varying the total number of points and the temporal range of the component sweeps as part of our analysis on the impact of temporal information to these networks. We achieve this by introducing two hyperparameters to their data loading pipeline: λ_{max} and λ_{rel}. When sampling a point cloud as input, we select λ_{max} total sweeps, and downsample the number of points in each sweep by the fraction λ_{rel} / λ_{max}, resulting in a point cloud that has as many points as one with only λ_{rel} distinct sweeps. When adjusting these hyperparameters, we denote the choices as (λ_{max} / λ_{rel}).

4 Methods

Initially we were interested in adapting the PointRCNN model [11] from OpenPCDet to train it on nuScenes. However, the specificity of tuning required in this domain and a lack of information on the logic behind existing hyperparameter choices meant we had little success in this approach. (See Appendix 7.1 for a visual comparison of the losses).

In addition, poorly documented code and a fragmentary codebase meant that resolving errors in evaluation was intractable with this architecture, and these errors inevitably continued to cause issues with future testing. Therefore, we shifted focus to simpler, pre-existing architectures already well-tuned for nuScenes. Particularly, we chose PointPillar-MultiHead. PoitnPillar learns the representation of a point cloud by organizing points into voxels in shape of vertical columns. Despite weaker performance metrics as drawn from OpenPCDet, its 3D backbone is a simple feature encoder that lends itself well to development.

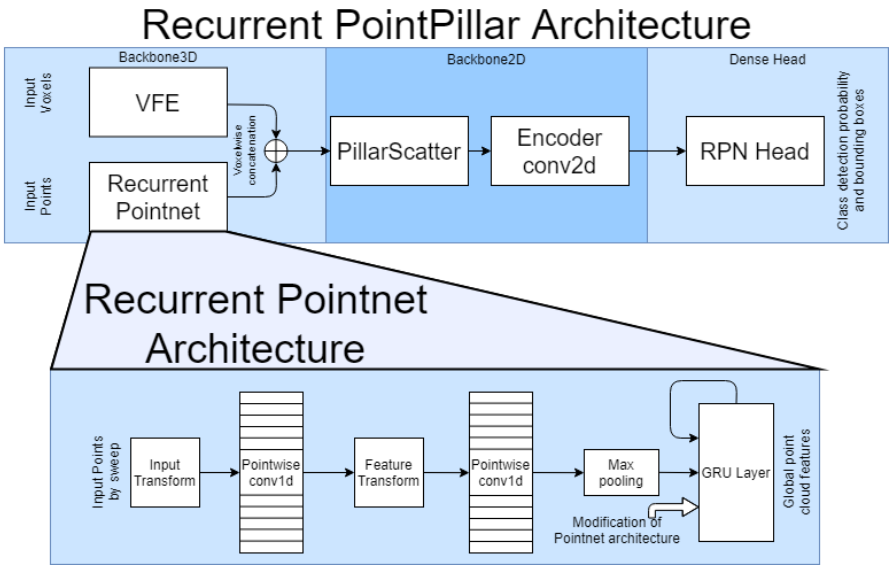
We extend this backbone by constructing a full PointNet architecture [6] to generate global point cloud features. PointNet is a simple architecture that can create spatial encoding; it uses a max pooling function as a symmetric function to destroy the ordering information to make the model insensitive to data permutation. In other words, we can input an arbitrary number of points to a PointNet, and it will output encoding for each point and also the entire point cloud simultaneously.

To connect the global features of sequential sweeps, we replace the final dense layers of PointNet with recurrent GRU layers, and then feed in consecutive sweeps. When performing semantic segmentation, PointNet appends the global information to each point; similarly, we concatenate the resulting global features to the voxel features created by the standard PointPillar backbone. To illustrate, suppose s_i is the i-th sweep, initial state of the hidden unit is $h_0 = \vec{0}$, and $\text{PointNet}(x) : \mathbb{R}^{n\times(3+C)} \to \mathbb{R}^m$ where $\mathbb{R}^m$ represents the global features, we have:

$$g_i, h_i = \text{GRU}(\text{PointNet}(s_i), h_{i-1}), i \in \{1, 2, ..., |S|\}$$

... and we will append global features $g_{|S|}$ to each of the pillar-like voxels.

The intuition here is that a recurrent network may be able to detect the continued presence of certain object classes such as motorcycles or bicycles, and alert the later layers to that class' presence in the point cloud through the global features. Below, we offer a diagram of our network on top of the base PointPillar network, using the same module-style diagrams that OpenPCDet displays.



5 Experiments and Discussion

5.1 Setup

For our experiments, we were limited rather significantly by the range of parameters we wished to explore and the computational cost of training on such large data inputs. As a result, we chose to explore 4 different combinations of (λ_{max}/ λ_{rel}): (5/5), (10/5), (15/5), and (10/10). The final choice (10/10) results in taking in point clouds equivalent to those processed by the pretrained models and serves as a comparison of compute power as well as the comparison of the effect of merely adding more points to the input, while the other three selections compare the effects of an ever increasing time window. We expect that if the time span grows too large, the base model's performance is likely to decrease, as the model will be working with outdated information. For our engineered model, we expect that it will still perform well even with older information due to the recurrent backbone.

Originally, we reused the hyperparameters provided by the original PointNet architecture. Due to memory limitations, we downsized the kernels and the output global features (from $\mathbb{R}^{1024}$ to $\mathbb{R}^{128}$). Both input and output feature size of the GRU are at least the same as the number of PointNet's global features as a smaller number may cause loss of global spatial information. We kept PointPillar's default parameter for output features (64), which means the each voxel will have 64 + 128 = 192 features before being fed into the 2D backbone.

For mini-batch sizing, we selected a batch size of 2 as this was the largest we could manage on the available hardware without running into memory issues. Besides, our recurrent PointNet requires an equal data size for all batches within a sweep, but the number of points sampled at a sweep can have huge variations across scenes/batches. To balance the input sizes across batches, we apply simple padding and downsampling for every sweep. Suppose p_{max} is the number of points in the largest batch and p_{min} is the number of points in the smallest batch. When $p_{max} \leq 2p_{min}$, we apply padding to any batch b with $p_b < p_{max}$, which means we duplicate (p_{max} - p_b) random points to make equal size; otherwise, we consider the difference between batches is too large and downsample any batch to p_{min} points to achieve equal-size batches.

All other parameters (e.g. NMS, anchors, regularization) were taken from OpenPCDet, under the assumption that they were well-selected for the base model and concatenation of extra features would not require a change to the learning rate.

5.2 Effects of temporal information on traditional networks

We trained the original PointPillar-Multihead model with the $\lambda_{max}/\lambda_{rel}$ combinations described above. Namely, this model merely takes in temporal information as an additional feature of a point.

$\lambda_{max}/\lambda_{rel}$	mAP	mATE	mASE	mAOE	mAVE	mAAE	NDS
Pretrained (10/10)	0.4463	0.3387	0.2600	0.3207	0.2874	0.2015	0.5823
(5/5) sweeps	0.1496	0.5275	0.4936	0.7162	0.7640	0.3220	0.2924
(10/5) sweeps	0.2221	0.4348	0.2945	0.6534	1.2613	0.3429	0.3385
(10/10) sweeps	0.2292	0.4919	0.2896	0.7514	0.7806	0.2749	0.3558
(15/5) sweeps	0.2652	0.4432	0.2750	0.5864	1.1728	0.2834	0.3738

We consider mAP, or mean Average Precision, as our primary metric, with perfect performance represented by a mAP of 1.0. The next 5 metrics are various errors we wish to minimize, with the final metric we wish to maximize devised by nuScenes as a weighted average of the others, labeled as "NuScenes Detection Score".

We can observe that the largest sampling window leads in most metrics, especially mAP. This result indicates that even treating temporal information as an extra feature can improve the performance of traditional networks. We did not observe a drop in performance with a larger window, meaning that we can increase λ_{max} for this model to achieve better metrics.

5.3 Effects of temporal information with our recurrent PointNet

Due to memory limitations, the global features are only calculated on subsamples from the entire set. A more powerful network might be able to achieve significantly more powerful global features

than our implementation. In addition, due to continued issues with the evaluation code for any model not out of the box, we were unable to actually obtain mAP or similar metrics for our own network, relying instead on the validation loss as our most useful obtainable metric.

$(\lambda_{max}/\lambda_{rel})$	(5/5)	(10/5)	(10/10)	(15/5)
Validation loss from traditional network (PointPillar)	-	-	-	.594
Validation loss from recurrent point cloud encoder	.872	.865	.792	.846

Again, we can observe some improvement with a larger sampling window. However, this may be attributed to the improvement from the traditional network. Also, the best performance does not come from the largest sampling window, indicating that the sampling window is too large for the recurrent encoder that is more sensitive to outdated data from earlier frames.

Besides, even though we were unable to obtain metrics from evaluation, we can still see that our recurrent PointNet has a worse training loss when both networks were trained for the same amount of epochs, which may indicate some flaws in the architecture. The fact that we used global features of the entire point clouds and even downsized those features due to memory limitations can contribute to an encoding method being too general for localization which will not help much in segmentation.

6 Conclusion and Future Work

Our experiments demonstrated that while keeping the input size the same, more spatiotemporal information can be effective in improving traditional object detection methods, and we suggest to treat the size of the sampling window as hyperparameters. The recurrent point cloud encoder does not seem to provide a convincing improvement to the detection performance, possibly due to global features being too general.

If we had more computational power and time, we would try to increase the number of global features output by our global in the recurrent network and use a more complex global encoder, PointNet++. Also, given that global features may not represent regional details well, we would like to investigate the possibility of splitting point cloud into partitions to run the recurrent PointNet on separately.

In addition, we would like to replace the GRU layer with an LSTM layer in our recurrent 3D encoder. Even though time only increases in its axis, the earlier frames of a sampling window may still contribute to the detection of the current state, which gives us the motivation to use LSTM. Training LSTM, however, takes much more time. Given our limited resources of computations (one 11GiB GPU), this time cost was unmanageable: while training one 1 epoch of our network using GRU takes 27 hours, training with LSTM would take 55+ hours.

Lastly, we want to explore furthest point sampling [12]. As mentioned in experiments, we downsampled our data size into 2500 points randomly. The furthest point sampling method chooses points uniformly distributed over each point cloud and covers the least known areas, which may give us a more optimal set of points.

Working with OpenPCDet's pipelines, while theoretically allowing us to move straight to model development and use pretrained hyperparameters, resulted in a significant amount of efforts to understand the structure of the code and debugging areas where it was likely not meant to be extended. Many of the issues encountered were never resolved, such as model results being produced only during training despite running on the same inputs as during testing (leading to our inability to visualize model outputs). Also, we had very limited access to computational resources, which means our debugging attempts and iterations were extremely time-consuming: with only one 11GiB GPU, training one epoch on the mini dataset took over 25 minutes and at least 27 hours on the full dataset. These challenges combined meant that much of our time was spent in areas that do not reflect on the improvement of our model, and perhaps indicates a lack of maturity or standardization in point cloud model development. While our results show promise, conclusive results will rely on the ability to dedicate significantly more development time, debugging time, and computing resources to testing these ideas than was available to our group.

7 Contributions

Code: Github Repository

Matei Armanasu: worked on data loading modifications, recurrent architecture development, and resolving issues with OpenPCDet.

Shrividya Manmohan: worked on cloud environment setup and running training and evaluation on EC2 instances and monitoring the training via Tensorboard.

Yuhao He: worked on the sweep processor & batch balancer, the recurrent PointNet with both GRU and LSTM, and optimization of GPU usage.

8 Acknowledgement

We are grateful to Davis Rempe for this project idea and valuable insights on modeling strategy and architecture design.

References

[1] Geiger, Andreas, et al. "Vision meets robotics: The kitti dataset." The International Journal of Robotics Research 32.11 (2013): 1231-1237.

[2] Yan, Yan, Yuxing Mao, and Bo Li. "Second: Sparsely embedded convolutional detection." Sensors 18.10 (2018): 3337.

[3] Lang, Alex H., et al. "Pointpillars: Fast encoders for object detection from point clouds." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

[4] Shi, Shaoshuai, et al. "From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network." IEEE Transactions on Pattern Analysis and Machine Intelligence (2020).

[5] Qi, Charles Ruizhongtai, et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space." Advances in neural information processing systems. 2017.

[6] Qi, Charles R., et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[7] Caesar, Holger, et al. "nuscenes: A multimodal dataset for autonomous driving." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020.

[8] Rempe, Davis, et al. "CaSPR: Learning Canonical Spatiotemporal Point Cloud Representations." Advances in Neural Information Processing Systems 33 (2020).

[9] Liu, Xingyu, Mengyuan Yan, and Jeannette Bohg. "Meteornet: Deep learning on dynamic 3d point cloud sequences." Proceedings of the IEEE International Conference on Computer Vision. 2019.

[10] Shi, Shaoshuai, et al. “Open-Mmlab/OpenPCDet.” GitHub, 2019, github.com/open-mmlab/OpenPCDet.

[11] Shi, Shaoshuai, Xiaogang Wang, and Hongsheng Li. "Pointrcnn: 3d object proposal generation and detection from point cloud." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

[12] Moenning, Carsten, and Neil A. Dodgson. Fast marching farthest point sampling. No. UCAM-CL-TR-562. University of Cambridge, Computer Laboratory, 2003.

Appendix

8.1 Adapting PointRCNN

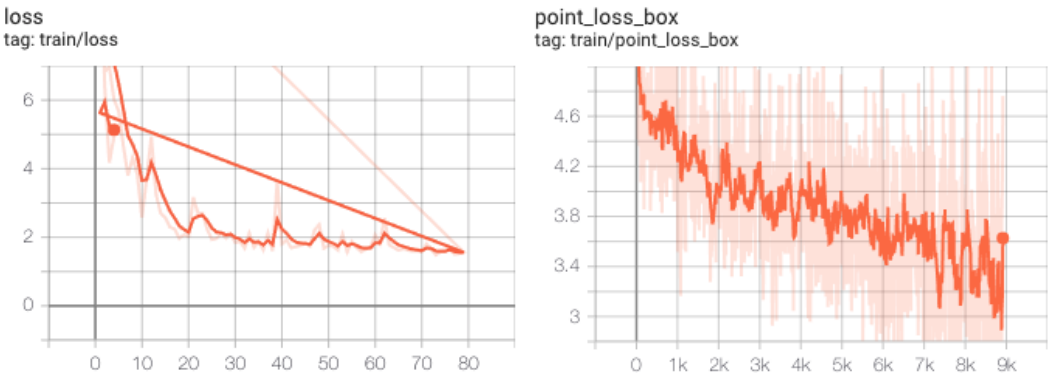


Figure 2: Left: loss curve of Second-MultiHead after 80 iterations (< 1 epoch); Right: loss curve of our adapted PointRCNN with adjusted learning rate and regularization after 9k iterations (65 epochs)