NPM3D PROJECT:

Object DGCNN: 3D Object Detection using Dynamic Graphs

Mehdi Zemni
mehdi.zemni@student-cs.fr

March, 2022

Abstract

3D object detection, one of the fundamental and challenging problems in autonomous driving, has received increasing attention from both industry and academia in recent years. Benefiting from the rapid development of deep learning technologies, 3D object detection has achieved remarkable progress. The objective of this project is to study the Object DGCNN: 3D Object Detection using Dynamic (Wang and Solomon [2021]) Graphs paper and to reproduce some of the experiments conducted by authors.

This paper proposes a method for object detection and velocity estimation in 3D LIDAR scans. It improves previous voxel- or pillar-based frameworks by adding a graph network and by avoiding non-maximum suppression by teaching the network to not produce duplicates through bipartite result-GT-matching. Additionally, the bipartite matching is used to design a knowledge destillation framework to create a further improved network.

The main contributions of this work is making a post-processing-free 3D object detection model achieving state of the art performance on autonomous driving benchmarks. The Object DGCNN paper is based on several prior works that motivated this approach and throughout this report I will explain these methods in detail.

1 Introduction

Most modern methods for object detection in 2D and 3D scenes include post-processing steps to achieve better performance. These methods will produce dense predictions with many overlapping bounding boxes and will refine predictions by reducing redundancy using post-processing techiniques like Non Maximum Suppression (NMS). These operations are typically non-parallelizable and inefficient even with modern deep learning frameworks.

The main objective of Object DGCNN paper is to introduce a new method with an NMS-free pipeline that achieves state of the art performance while running faster. This is achieved using local point cloud features on a birds eye view grid collected using either PointPillars Lang et al. [2018] which maps sparse point clouds onto a dense BEV pillar map on which we can apply 2D convolutions or SparseConv Graham et al. [2017] to perform 3D convolutions on voxel wise features and then averaging along z-axis to obtain BEV 2D features. Then comes DGCNNs Wang et al. [2018] which are conventionally used for point cloud classification and segmentation. In their work, authors generalize DGCNN to model objects as a point set and

use a permutation-invariat loss to measure the error between predicted and ground-truth set of objects.

The second contribution of this paper which is a consequence of the previous one, is a set-to-set distillation method for 3D object detection. In past 3D object detection methods, since final predictions are heavily processed using NMS, distilling the teacher knowledge to the student network was not efficient and didn't yield good results.

In what follows, we will describe the object DGCNN pipeline step by step in more details that what was described in the paper. Some of the steps are existing methods that were proposed in prior works and authors didn't explain in detail those steps.

2 Feature extraction

The first step of object DGCNN is to collect bird-eye-view features using either PointPillars Lang et al. [2018] or SparseConv Graham et al. [2017]. These BEV features will be used later to predict a set of bounding boxes.

2.1 PointPillars

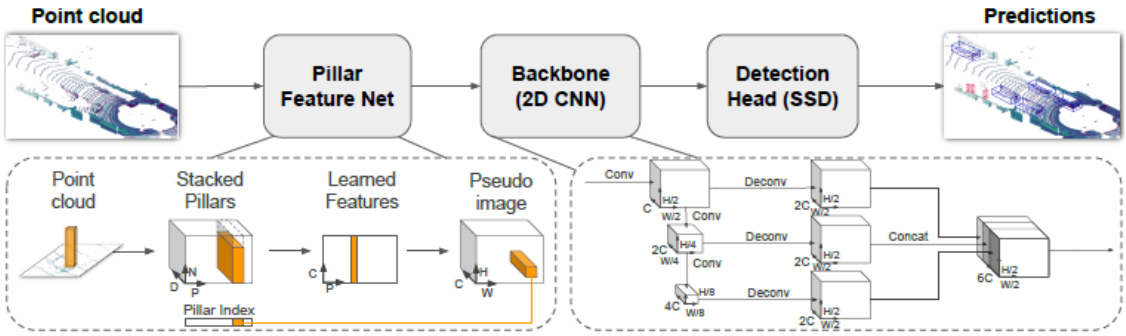


Figure 1: PointPillar Network overview.

PointPilliars is a method introduced to estimate oriented 3D bounding boxes for cars and pedestrians. The network of PointPillars consists of three main stages (Figure 1): (1) A feature encoder network that converts a point cloud to a sparse pseudoimage; (2) a 2D convolutional backbone to process the pseudo-image into high-level representation; and (3) a detection head that detects and regresses 3D boxes. In our pipeline we will only use the first two components since detections will not be directly performed on PointPillar features. This method uses 2D convolutions which makes it very efficient and fast during inference. To apply 2D convolutions, we need first to transform the point cloud to a pseudo image. As its name suggest we will create pillars on the ground plane (x-y plane). First, the point cloud is discretized into an evenly spaced grid in this plane, creating a set of pillars $\mathcal{P}$ with $|\mathcal{P}| = B$. Each point in every pillar will be described using a $D = 9$ dimensional vector:

- Point coordinates x, y, z and reflectance r
- x_c, y_c, z_c distance to the the arithmetic mean of all points in the pillar

- x_p, y_p distance to the center of the pillar

Since most pillars will be empty or sparse we will impose a limit on the number of pillars per sample (P) and a limit on the number of point per pillar (N) in order to create a fixed size tensor (D, P, N). If a sample or pillar holds too much data to fit in this tensor the data is randomly sampled and if a sample or pillar has too little data to populate the tensor, zero padding is applied. After that, a small PointNet Qi et al. [2016] is applied to this point cloud of $D \times N$ points to generate a (C, P, N) pointcloud. This is followed by a max operation over the channels to create an output tensor of size (C, P). Recall that P is the number of pillars per samples. The final step of encoding will consist in recovering the 2D information (original position of pillars) by creating a pseudo image of size of size (C, H, W).

Finally, to process these pseudo-image into high-level representation we use a 2D convolutional backbone.

2.2 SparseConv

An alternative BEV embedding is SparseConv Graham et al. [2017]. Instead of forming pillars that contain a number of points and then using PointNet to obtain Pillar-wise features we use voxels and we collect voxel-wise features. In contrast to PointPillars, SparseConv conducts 3D sparse convolutions to refine the voxel-wise features. Finally these features are averaged along the z axis to form pillar-wise features resulting in a pseudo image like before.

3 Dynamic Graph CNN

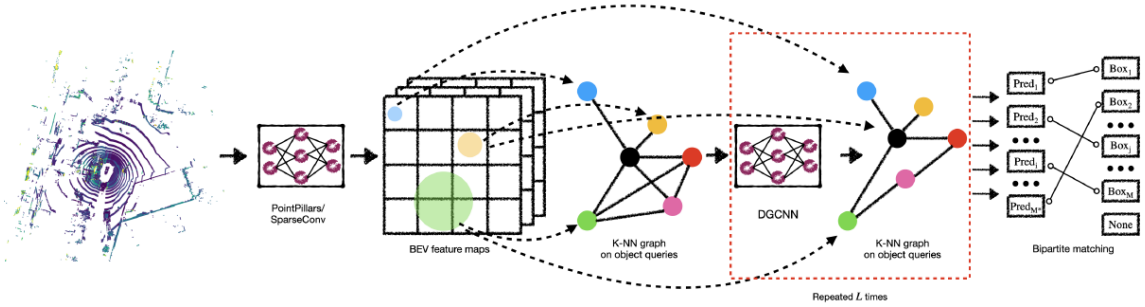


Figure 2: Overview. Point cloud features are learned in BEV, followed by L DGCNNs to model object relations. Model predicts a set of bounding boxes and a set-to-set loss is computed.

This paper prposes a NMS-free method for object detection, following the 2D representative work DETR Carion et al. [2020]. But instead of using Transformers, it uses a dynamic graph CNN (DGCNN) Wang et al. [2018] to generate the set of detected objects. Unlike graph CNNs, DGCNN is not a fixed graph but it is dynamically updated after each layer of the network by using the k-nearest neighbors of points computed from the sequence of embeddings. In conventional graph CNNs, we use proximity in the input space which is fixed during training and through all layers of the network.

To perform NMS-free detections, a DGCNN like network is used to perform a series of set-based computations to produce bounding box predictions from the BEV feature maps.

Each layer l of object DGCNN operates on a set of queries $Q_l = \{q_{l1}, ..., q_{lM*}\}$ and predicts Q_{l+1}. These queries represent object positions that are progressively refined at each layer. Q_0, the initial queries, are also learned during training, yielding a dataset-specific prior.

Now, we will present the operations performed at a layer l. Following the notations of the paper, we start with queries $Q_l = \{q_1, ..., q_{M*}\}$. Let's consider a single query $q_i \in R^Q$. We use 3 sub neural networks Φ_{ref}, Φ_{neighbor} and Φ_{atten} that share weights among queries and perform the following operations:

$$p_i = \Phi_{\text{ref}}(q_i) \tag{1}$$

$$\{\delta_i^0, ..., \delta_i^k, ..., \delta_i^K\} = \Phi_{\text{neighbor}}(q_i) \tag{2}$$

$$\{w_i^0, ..., w_i^k, ..., w_i^K\} = \Phi_{\text{atten}}(q_i) \tag{3}$$

Where $p_i \in R^2$ are reference points , $\{\delta_i^0, ..., \delta_i^K\} \subset R^2$are the offsets, and $\{w_i^0, ..., w_i^K\}$ are the attention weights. p_i's can be regarded as centers of bounding boxes and the δ's as the positions of K key-points that determine the position and the geometry of the object. Next, we collect a BEV feature f_i^k associated to each neighbor point $p_i^k = p_i + \delta_i^k$

$$f_i^k = f_{\text{bilinear}}(\mathcal{F}^d, p_i + \delta_i^k) \tag{4}$$

where f_{bilinear} bilinearly interpolates the BEV feature map $\mathcal{F}^d$. Then, a single object query feature f_i is aggregated from the f_i^ks using the learned attention weights as follows:

$$f_i = \sum_k \text{softmax}(w_i^k) f_i^k \tag{5}$$

Until here, we did not incorporate yet neighborhood interactions between q_is. This can be modeled using DGCNN to capture a sparse set of relations. Neareast neighbors of each query q_i are computed in each layer in the feature space yielding 16 query neighbor for each query. We obtain a graph of $M*$ vertices representing the $M*$ queries $Q_l = \{q_1, ..., q_{M*}\}$ and each node i is modeled with the previously computed features f_i. Each node is connected to itself and to its 16 neighbors in a directed way. Identically to DGCNN, we learn a feature per edge e_ij and then aggregate back to the vertices i to produce the new set of object queries. This can written as:

$$q_i^{(l+1)} = \max_{\text{edges } e_ij} \phi_{\text{edges}}(f_i, f_j) \tag{6}$$

where max denotes a channel-wise maximum and ϕ_{edges} is a neural network for computing edge features. To sum up, we first compute features f_i of each query point q_i and then we apply DGCNN layer on features f_i to get the point set $\mathcal{Q}_{l+1}$ and we repeat these steps at each layer until we get $\mathcal{Q}_L$ where L is the total number of layers. Finally, each query q_i^L goes through a classification network to predict its label $\widehat{c}_i$ and a regression network to predict its bounding box parameters $\widehat{b}_i$.

4 Loss

The last task of object-DGCNN is to assign predictions to ground truth bounding boxes and to compute a set-to-set loss. Traditional object detection models uses a combination of regression and classification loss as follows:

$$\mathcal{L} = \sum_{j \in \text{BEV pixels}} \mathcal{L}_{cls}(\widehat{c}_j, c_{\widehat{\sigma}(j)}) + \lambda 1_{\{c_{\widehat{\sigma}(j)} \neq \emptyset\}} \mathcal{L}_{reg}(\widehat{b}_j, b_{\widehat{\sigma}(j)}) \tag{7}$$

$$= \sum_{j \in \text{BEV pixels}} -\log \widehat{p}_{\widehat{\sigma}(j)}(\widehat{c}_j) + \lambda 1_{\{c_{\widehat{\sigma}(j)} \neq \emptyset\}} \left\| \widehat{b}_j - b_{\widehat{\sigma}(j)} \right\|_1 \tag{8}$$

where $\widehat{\sigma}(.)$ is a function that assigns each prediction (on the BEV pixel j) to a ground truth bounding box, $\log \widehat{p}_{\widehat{\sigma}(j)}(\widehat{c}_j)$ is the log probability of the j^{th} prediction associated to class $c_{\widehat{\sigma}(j)}$, the class of the associated ground truth bounding box $\sigma(j)$. An additional class is created $= \emptyset$ which corresponds to predictions with no associated ground truth. We include these predictions (where $c_{\widehat{\sigma}(j)} \neq \emptyset$) in the cross entropy loss to learn to classify invalid bounding boxes but these are not included in the regression loss since we are not interested in their corresponding exact coordinates. Different matching functions $\widehat{\sigma}(.)$ can be used but the most simple function that is commonly used like in Pillar-OD Wang et al. [2020] and CenterPoint Yin et al. [2020]:

$$\widehat{\sigma}_{\text{overlap}}(j) = \begin{cases} k & \text{if } b_k \text{ overlaps with Bev pixel} j \\ \emptyset & \text{otherwise} \end{cases} \tag{9}$$

This strategy is used in dense object detectors that perform dense predictions (prediction per BEV pixel). This is similar to anchor based object detection methods used in 2D. Since the use of this objective function coupled with the above assignment function will encourage each BEV pixel to predict the same surrounding box, this will yield many redundant bounding boxes. This is why confidence aggregation or non-maximum suppression post-processing is necessary.

Unlike these dense prediction methods, Object-DGCNN predicts a sparse set of bounding boxes as explained in the previous sections. After that, authors propose to match the prediction set with the ground-truth set in a one-to-one fashion using an optimal bipartite matching and finally to evaluate a set-to-set object detection loss. The optimal matching function is defined as follows:

$$\sigma^* = \arg\min_{\sigma \in \mathcal{P}} \sum_{j \in \text{detections}} -\log \widehat{p}_{\sigma(j)}(\widehat{c}_j) + \lambda \left\| \widehat{b}_j - b_{\sigma(j)} \right\|_1 \tag{10}$$

using the same notations as before. The only difference is that here we sum over a sparse set of detections, still M^*, the fixed number of predictions per sample, is much larger than M, the number of ground truth boxes. Therefore, there will be $M^* - M$ unmatched predictions which will be assigned an invalid class ($c_{\sigma^*(j)} = \emptyset$ class). This is a typical assignment problem in a bipartite graph of M^* and M nodes that can be solved efficiently using the hungarian algorithm Kuhn [1955] to find the optimal assignment function $\sigma^*(.)$.

Finally the set-to-set loss is computed as follows:

$$\mathcal{L} = \sum_{j \in \text{detections}} \mathcal{L}_{cls}(\widehat{c}_j, c_{\sigma^*(j)}) + \lambda 1_{\{c_{\sigma^*(j)} \neq \emptyset\}} \mathcal{L}_{reg}(\widehat{b}_j, b_{\sigma^*(j)}) \tag{11}$$

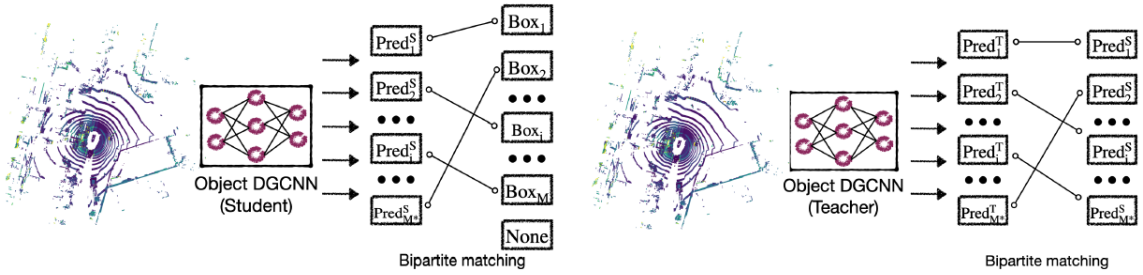
$$= \sum_{j \in \text{detections}} -\log \widehat{p}_{\sigma^*(j)}(\widehat{c}_j) + \lambda 1_{\{c_{\sigma^*(j)} \neq \emptyset\}} \left\| \widehat{b}_j - b_{\sigma^*(j)} \right\|_1 \tag{12}$$

Now we have the two ingredients authors introduced to design a post-processing free 3D object detector.

- The use of DGCNN to model object-object interactions so objects have information about their peers.
- The set-to-set loss which enforces the model to learn to reduce redundancy. My intuition is that the network learns the NMS and produces a single high-confidence prediction per object.

Since a fixed number of predictions is used during inference, it is possible that these might include duplicate predictions, though with a lower confidence score.

5 Distillation



(a) Student network (b) Teacher network (fixed)

Figure 3: The set-to-set distillation pipeline. The student network is trained with the ground-truth supervision as well as with the supervision from a fixed teacher network.

As designed by authors, this model enables a simple approach to distillation via set-to-set distillation. This distillation is not trivial using previous architectures such as anchor-based prediction or pillar-based prediction.

Although the the distillation method is easy to understand and extremely straightforward to implement with this set-to-set sparse structure, authors didn't explain well the connection between NMS-free object detection and distillation. I couldn't understand clearly its utility in NMS-free detection and how it is supposed to improve results especially when the teacher and the student are identical and take the same point clouds as input.

First, let's see how the distillation is performed. Knowledge distillation extracts general knowledge from a pre-trained teacher network and provides guidance to a target student

network. Most common methods to distill knowledge include feature distillation which enforce intermediate features of the student network to lie close to teacher features. Another common method is the pseudo label based distillation which consists in generating pseudo training examples with the pre-trained teacher network to train the student.

In the object-DGCNN paper, authors propose a set-to-set distillation method which consists in aligning the last stage of the object detection (detection head) of the student network with its counterpart in the teacher network using an optimal matching function σ_d^* between the output set of the teacher and that of the student. the matching function is defined as follows:

$$\sigma_d^* = \underset{\sigma_d \in \mathcal{P}}{\arg\min} \sum_{j \in \text{detections}} -\log p_{\sigma_d(j)}(c_j^{\mathcal{T}}) + \lambda \left\| b_j^{\mathcal{T}} - b_{\sigma_d(j)}^{\mathcal{S}} \right\|_1 \tag{13}$$

where $c_j^{\mathcal{T}}$, $b_j^{\mathcal{T}}$ and $c_j^{\mathcal{S}}$, $b_j^{\mathcal{S}}$ are the corresponding outputs of the teacher and student models. The loss used to train the student model is a combination of two losses $\mathcal{L} = \alpha\mathcal{L}_{\text{det}} + \beta\mathcal{L}_{\text{distill}}$: the first one is the supervised loss as defined before (11) and the second loss is the distillation loss (supervision from teacher) which is the same as the detection loss but in which we replace ground truth bounding boxes c_j, b_j with teacher outputs $c_j^{\mathcal{T}}$, $b_j^{\mathcal{T}}$ and σ^* with σ_d^*.

Depending on the feature extractor network (PointPillar or SparseConv), the DGCNN model yields different results after training in terms of speed and accuracy. For instance, object DGCNN with the SparseConv backbone achieved a 5.5% improvement in terms of mAP (mean average precision) over the object DGCNN with the PointPillar backbone according to the paper results but it runs slower because it uses 3D convolutions instead od 2D convolutions.

To improve the PointPillar network without using a larger network, we can distill knowledge from a pretrained Voxel based network (teacher) to the Pillar based network (student). Another application of distillation is to distill information from a teacher network pretrained with privileged information. For example, we use a teacher trained on dense point clouds, and distill information to train a student network that has access to sparse point clouds only.

6 Experiments

The implementation of this paper is built on top of the MMDetection3D Contributors [2020] package. Unfortunately, authors do not provide a clear requirement file to create a stable environment with the correct versions of packages. After many attempts and by looking at the logging files of pretrained models, I managed to find most of the packages correct versions. However, I had to change the visualisation code since I had conflicts of versions with open3D package. I also provide a docker file to build an image identical to my environment.

I used the pre-trained models provided by the authors and tested them on the nuScenes dataset Caesar et al. [2019] (6). Test results are summarized in table 1. Test results were coherent with results presented in the paper but the inference time was not expected since I registered a longer inference time with the pillar network (although it has less parameters). I also adapted some code from nuScenes github repository to visualize some test results. In fig.4 we show predictions and annotations of bounding boxes in bird eye view. We can also show results in 3D in CloudCompare (fig.5).

Another experiment that would have been interesting to perform is to train the model on other datasets such us KITTI dataset Geiger et al. [2013]. Unfortunately, I didn't manage to do this experiment because of the size of training sets.

Backbone	NDS ↑	mAP ↑	Inference time †
Pillar	62.8	53.2	745 s
Voxel	66.0	58.6	614 s

Table 1: Test results on nuScenes dataset († on RTX2080).

7 Conclusion

Though none of the building blocks used in the paper is novel (PointPillars and SparseConv as backbone, DGCNN for graph processing, DETR for the NMS-free bipartite matching, and a common student-teacher knowledge destillation), the combination is non-trivial and well thought off. The knowledge destillation would not be possible that easily without the NMS-free architecture.

Although the method doesn't yield the best results when compared to state of the art methods it might open up additional research on NMS-free object detectors.

References

Yue Wang and Justin Solomon. Object DGCNN: 3d object detection using dynamic graphs. *CoRR*, abs/2110.06923, 2021. URL https://arxiv.org/abs/2110.06923.

Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *CoRR*, abs/1812.05784, 2018. URL http://arxiv.org/abs/1812.05784.

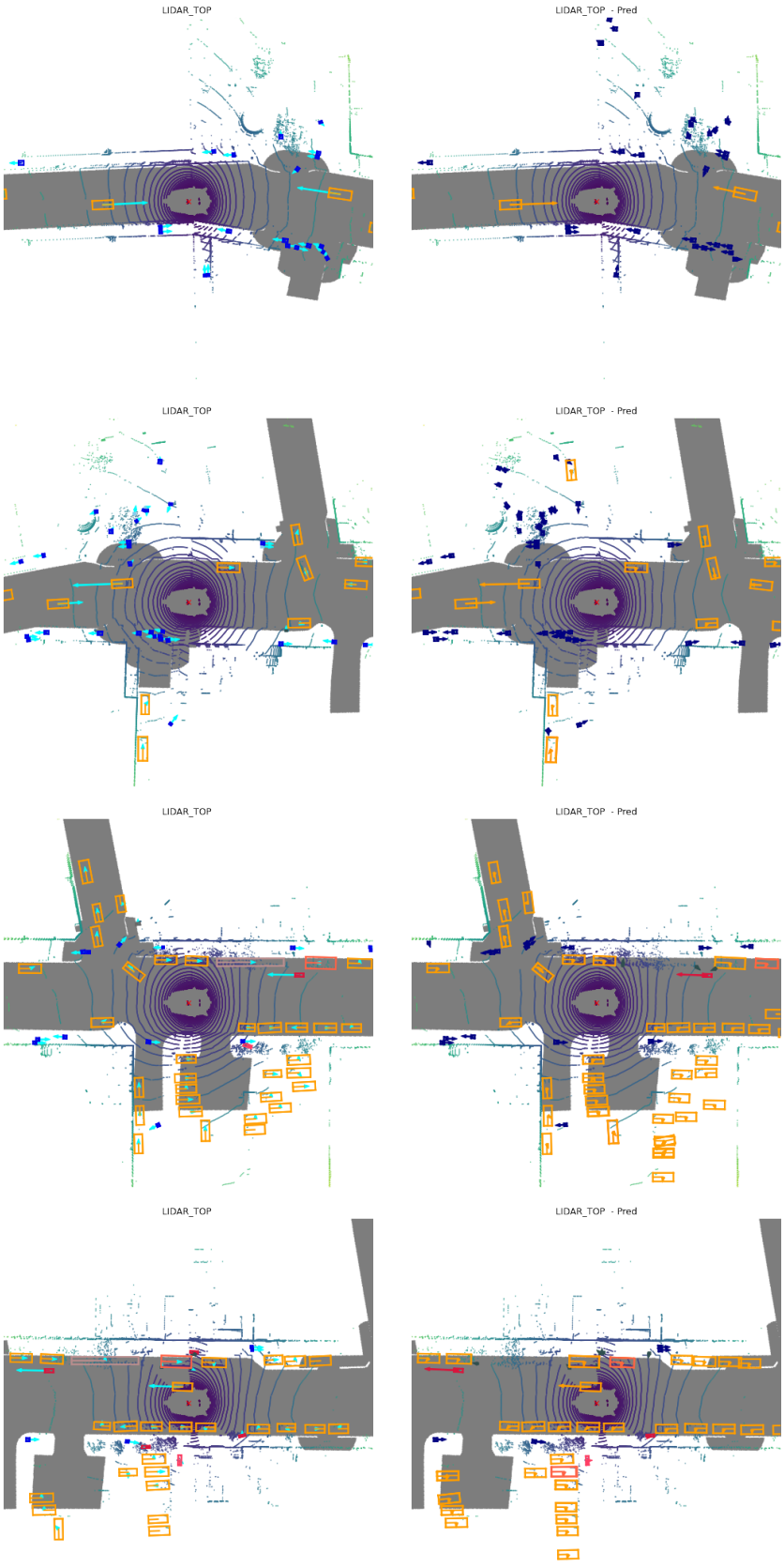
Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. *CoRR*, abs/1711.10275, 2017. URL http://arxiv.org/abs/1711.10275.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018. URL http://arxiv.org/abs/1801.07829.

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016. URL http://arxiv.org/abs/1612.00593.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *CoRR*, abs/2005.12872, 2020. URL https://arxiv.org/abs/2005.12872.

Yue Wang, Alireza Fathi, Abhijit Kundu, David A. Ross, Caroline Pantofaru, Thomas A. Funkhouser, and Justin Solomon. Pillar-based object detection for autonomous driving. *CoRR*, abs/2007.10323, 2020. URL https://arxiv.org/abs/2007.10323.



Ground truth 9 Prediction

Figure 4: Predictions using Object-DGCNN with PointPillars backbone

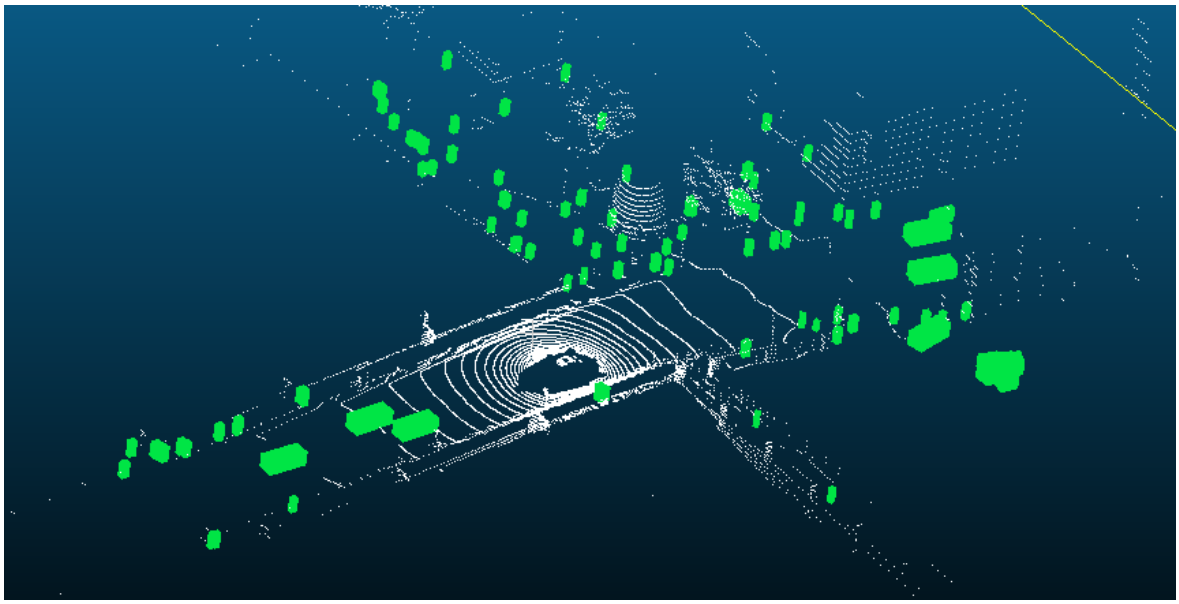


Figure 5: Predictions using Object-DGCNN with PointPillars backbone.

Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3d object detection and tracking. *CoRR*, abs/2006.11275, 2020. URL https://arxiv.org/abs/2006.11275.

Harold W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, March 1955. doi: 10.1002/nav.3800020109.

MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. https://github.com/open-mmlab/mmdetection3d, 2020.

Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

https://phortail.org/club-informatique/definition-informatique 136.html. Date de dernière consultation : 26/08/2019.

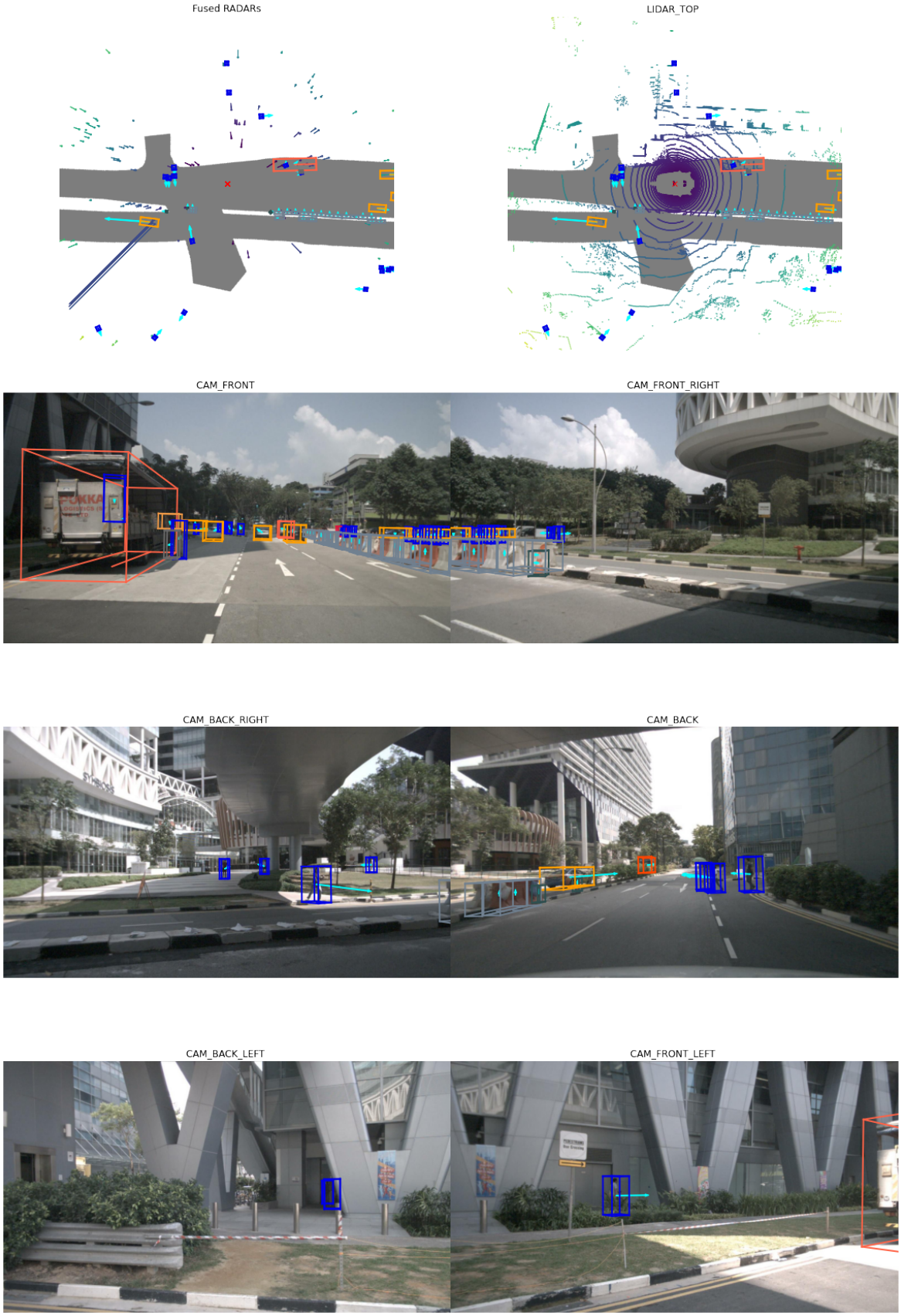


Figure 6: Sample from nuScenes dataset.