Extended Graph Convolutional Networks for 3D Object Classification in Point Clouds

Sajan Kumar[1], Sai Rishvanth Katragadda[2], Ashu Abdul[3], V. Dinesh Reddy[4]
School of Engineering and Sciences
Department of CSE, SRM University,
Amaravati, AP, India

***Abstract*—Point clouds are a popular way to represent 3D data. Due to the sparsity and irregularity of the point cloud data, learning features directly from point clouds become complex and thus huge importance to methods that directly consume points. This paper focuses on interpreting the point cloud inputs using the graph convolutional networks (GCN). Further, we extend this model to detect the objects found in the autonomous driving datasets and the miscellaneous objects found in the non-autonomous driving datasets. We proposed to reduce the runtime of a GCN by allowing the GCN to stochastically sample fewer input points from point clouds to infer their larger structure while preserving its accuracy. Our proposed model offer improved accuracy while drastically decreasing graph building and prediction runtime.**

***Keywords*—Object classification; graph convolution networks; non-autonomous driving**

I. Introduction

Autonomous vehicles are becoming the future of mobility, supported by advances in deep learning techniques. Point cloud learning has lately attracted increasing attention due to its wide applications in many areas, such as computer vision, autonomous driving, and robotics. Recent advances in graph convolution networks suggest that graph representations could provide better features for point cloud processing. Graphs are one of the most common data structures in the analysis and storage of the real-world data-modeling of social networks, roads, etc. However, the amount of work devoted for developing the neural network models to process graphs has not been proportional to the amount of data available for such analysis. In the past couple of years, some researchers have looked at generalizing current neural network models to process arbitrary graphs, some of which we will briefly summarize later in this section. Thomas et al. [1] offers a brief overview of GCNs and the architecture is shown in Fig. 1. In essence, most GCNs have a universal architecture and their convolutional nature arises from sharing filter parameters over all graph locations.

The objective of a GCN is to learn a function of features for a given graph G, which takes the inputs:

- X, an $N \times D$ feature matrix which summarizes a feature description x_i for every node i (where N is the number of nodes, and D the number of input features),
- A descriptor of the graph's structure in matrix form, usually as an adjacency matrix A.

This produces an output Z, which is a node-level $N \times D$ feature matrix (F is the number of output features per node).

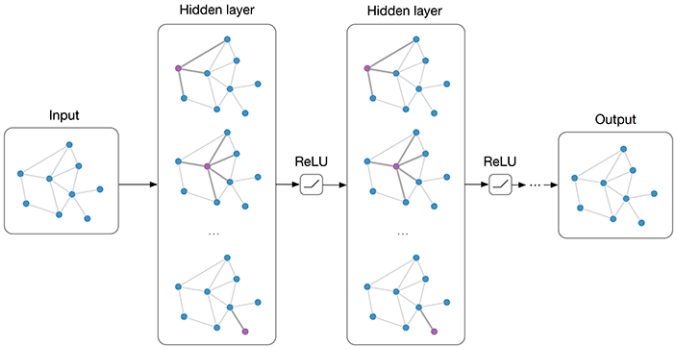

Fig. 1. Overview of a Graph Convolutional Network (GCN) and its Associated Layers [1].

Each layer H^l out of the total layers L in the GCN can then be modeled as a nonlinear function:

$$H^{(l+1)} = f(H^{(l)}, A)$$

, where

$$H^{(0)} = X \text{ and } H^{(L)} = Z. \quad (1)$$

Most GCNs only differ in how the nonlinear function f is parametrized and chosen.

The rest of this paper is organized as follows. Section II presents motivation as well as related works. Limitations and contributions are presented in Section III. Section IV elaborates the pointGCN model and the proposed work is presented in Section V. Experimental studies are then presented in Section VI, followed by a discussion on the results in Section VII. Finally, conclusions are provided in Section VIII.

II. Related Works

Existing research on processing the arbitrary graphs proposed the usage of the neural networks for extracting the information from the arbitrary graphs. Bruna et al. [2] used the graph-based analogs with the convolutional neural networks (CNN) to obtain an efficient architecture by reducing the number of parameters, relying on hierarchical clustering of the graph and by analyzing the spectrum of the graph's Laplacian matrix. David et al. [3] introduced a CNN approach with an end-to-end pipeline that could operate directly on arbitrarily sized graphs to generalize molecular feature extraction. Jain et al. [4] used the high-level modeling ability of spatio-temporal graphs to improve a recurrent neural network architecture to model the sequential computer vision tasks (like human or

object interactions) more effectively. Mich et al. [5] proposed a true generalization of CNNs to work on any graph structure by developing a model based in spectral graph theory, which affords the same linear computational and learning complexity as traditional CNNs. This work constitutes a big jump forward in the field in terms of modeling GCNs. Moreover, Kipf et al. [1], created a GCN model based on spectral graph convolutions that scale linearly in the total number of graph edges. The model layer representations encoded the specific features of nodes as well as the local structure of the graph around a given region. Hence it is clear that GCNs offer a better scheme for the analysis of data that is generally arbitrarily grouped or sparse in nature, such as the structure seen in point clouds. In terms of the application of CNNs to the processing of 3D point clouds, PointNet [6] is the first major DL method for 3D classification and segmentation. The network directly takes point clouds as input and allows for efficient and effective classification, segmentation and scene semantic parsing. The model proposed by kiran et al. [7] uses 3D prior maps to reduce the computational requirements on 3D point clouds. Zar zar t al. [8] proposed the PointRGCN for using the point cloud inputs for vehicle tracking. The model uses a residual GCN and a contextual GCN for refining a 3D object from a point cloud using a graph representation of the object.

We propose to extend the PointGCN model developed by zang et al. [9]. We down sample the graph with localized convolutions to obtain the latent features for describing the local structures of the input point cloud. Our approach attempts to leverage the flexibility of GCNs in dealing with unstructured input with the inherent nature of point clouds. From our experiments, we see that proposed model reduces the computational requirements on 3D point clouds and improves the performance.

III. Current Limitation of GCN Models

Most of the research in the field of point cloud computation involves converting of 2D image to a point cloud to extract information. The existing research focusses on utilizing the depth information collected via a LiDAR for classification or segmentation (as done in PointNet). We know that such methods are accurate individually, but do not necessarily make the best use of the data at hand. The GCN models which we reviewed are effective but not necessarily specialized for real-world object classification tasks. PointGCN [9] does not appear to target any specific application and hence is not specifically optimized for any purpose. Likewise, PointRGCN [8] mainly works on bird's-eye view detection and classification tasks rather than typical views from a camera on the ground. It also uses a variety of different models to obtain its final result. Through our work, we would like to extend [9] and [8] to fit other kinds of point cloud datasets without restricting to autonomous driving. We will be looking into using some of the datasets at [10] to train on and will attempt to generalize the model to gain good predictive performance on these varied datasets.

The main contributions of the paper are:

- Extending and generalizing a GCN model to work on other kinds of LiDAR point clouds such as objects in urban environments or the interiors and exterior of buildings, without suffering a significant performance loss.
- Optimizing and generalizing current GCN model such that it can run on small computing processors such as arm cortex A7.

Our work will attempt to combine useful features from the current state of the art (for 3D object classification) into an innovative approach that will hopefully match current performance in the field at a reduced time cost. The models currently used in research (processing on a point cloud with PointNet and RGB feature/depth extraction) have their own merits and demerits. 2D RGB images are better for feature extraction and segmentation at a reduced time cost. However, the 2D RGB images lack the depth information, which leads to poor capturing of the relationship between the subject and the objects. In our approach we use the point clouds to bridge this gap for extracting the relationship between the objects and the subject. We note that we can generalize the current methods in the field to adapt with different types of data. In our approach, we adapt a method without suffering the feature loss. This method will be important for the researchers working on autonomous driving, cave mapping, home modeling, and video game designers, to name a few.

IV. Architecture of the PointGCN Model

The PointGCN appears to have the potential for flexibility in inputs and did not require the processing of its data through multiple models [9]. It was also extremely lightweight in terms of the code involved, and so would be easier to modify.The following is a brief overview of the model architecture for a Graph Convolutional Network for the purpose of 3D object classification as used in [9]. Similar to most simple CNNs, the GCN consists of a convolution layer, a pooling layer, and a fully connected layer. The graph convolutional layer allows the GCN to encompass the structural information of the object to be able to discriminate between them. Graph Laplacians of the input point cloud data are generated. The graphs are normalized to keep them to a uniform spectrum range. As described in [11], the ChebyNet Graph-CNN performs better on homogenous graphs for prediction tasks such as image classification. Hence, a similar transformation is applied to the heterogeneous graphs generated from point clouds. To obtain a single level of a feature transformation, the model applies Chebyshev polynomial filters, which keep the learned feature maps localized.

A Rectified Linear unit (ReLu) nonlinear activation function is applied at the output of each graph convolutional layer. Furthermore, global max pooling is performed after these activations are applied. This allows for the computation of the global statistics of all the output points. When multiple graph convolutional layers are used, the statistics of each layer are used together for the final probability computations.

The version of the model we used two graph convolutional layers, and the Chebyshev polynomials are of order = 4 and order = 3 for the two layers respectively. The rest is similar to as used in [9], with a 40-nearest neighbor graph for each point cloud object for graph convolution and global pooling. The intermediate outputs of the graph convolution layers are passed through a final fully-connected linear layer

with softmax activation as a flattened vector. The number of output nodes in the output layer corresponds to the number of labels in the dataset, which in our case, we kept at 40 labels. Hence, the output is a vector where each index corresponds to the probability of the input belonging to that label.

V. Proposed Approach - Modified PointGCN

PointGCN was geared to be used with the ModelNet40 dataset. Therefore, in order to use the model with different datasets and different labels, we had to make the model label-agnostic, and make sure the data format we wanted to use worked everywhere in the model.

We also approached the problem from another angle. The initial implementation in [9] used 2048 point cloud examples per test/train set, and each point cloud consisted of 2048 points. This amounted to an extremely large amount of input data and meant the creation of the nearest-neighbor graphs from the point cloud data was computationally expensive and extremely memory-intensive. It often formed a bulk of the actual runtime of the network where data was fed into it. Therefore, we tried running the model and testing its accuracy with fewer training examples and points per cloud instead, which would significantly reduce runtime if successful.

We then also attempted to make some changes in the PointGCN model itself, by introducing link prediction and an entity classification layer, which helps in the recovery of missing points or facts from the dataset, and recovery of attributes corresponding to the missing entity, which was motivated by work in [12]. Hidden representation of GCN of ith layer is computed by:

$$h_i^{l+1} = \sigma\left(\sum_{jN_i} \frac{1}{c_i} W^l h_j^{(l)}\right)$$

If we introduce the link prediction and entity layer in the above function to utilize this hidden representation we can compute the edges more freely by:

$$h_i^{l+1} = \sigma\left(W_0^l h_i^l + \sum_{rR} \sum_{jN_i} \frac{1}{c_{i,r}} W_r^l h_j^{(l)}\right)$$

where $N_{r,i}$ denotes the set of neighbor indices of node i under relation rR and ci,r is a normalization constant. In the entity classification layer the GCN uses $c_{i,r} = |N - ri|$. This method helps determine a node's relational encoding, aids in nearest-neighbor finding, and helps reduce the computation cost. Though the there is an increase in the number of hyperparameter's the paper [13]

We decided to try and generalize the PointGCN model to achieve a good accuracy over two different datasets:

- Princeton ModelNet40 Dataset [14]: A volumetric representation of 3D objects across 40 categories, generated in a highly controlled environment using CAD models.
- CMU Oakland Dataset [15]: Contains 44 labels as shown in Fig. 2. This is generated using a SICK LMS laser scanners and used in push-broom, collected around the CMU campus in the neighborhood of Oakland, Pittsburgh, and contains X, Y, and Z point coordinates, corresponding labels, and confidence as properties.

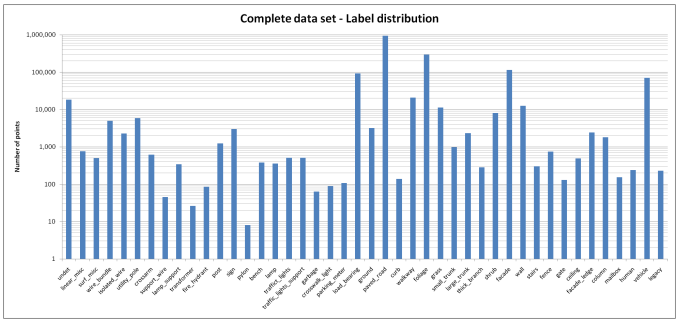

Fig. 2. A Log-scale Plot of the Distribution of the 44 Different Labels in the CMU Oakland dataset. The Four Labels with the Lowest Frequencies were Dropped out to Maintain Consistency with the 40 Outputs in the ModelNet40 Dataset.

A. Preparing Input Datasets

We faced some challenges in deciding how to generalize the input datasets to the model. The biggest is the representation of the data points in point clouds. Each prepared dataset uses a different representation of 3D point clouds, which is specific to a single model's architecture. Furthermore, the collection methods and reference frame of the collecting device also vary across datasets. Some datasets contain far more information (such as semantic annotations or segmentations) that cannot be found in other datasets. Each dataset also has a different density and distribution of points for every object in the set. The CMU Oakland dataset has labels for objects that occur so infrequently in the set that the data is unusable on a larger scale (mainly power cables or wires in the background). The PointGCN model constructs its initial graph data from two arrays found in an HDF5 file. The first, the 'data' array, is an N-by-M array with N point cloud examples and M points per point cloud. Each point in the point cloud is a 3-by-1 array consisting of the X, Y, and Z coordinates of the point. The second array is the 'label' array, an N-element array, with each index corresponding to a point cloud in the 'data' array. This meant there must be a generalized method of preparing or feeding the respective data into the model. As such, for object classification, we decided to use the bare minimum. A preprocessed dataset to the model should contain the X, Y, and Z coordinates per point, along with the object label to which that point belongs.

B. Dataset Creation

To overcome the issues we faced with point sparsity and the parsing of datasets, we wrote a script to process and 'mass-create' datasets that we could test on. This script was tested and used on the CMU Oakland dataset [15] which is shown in Fig. 3. It requires the point cloud x, y, and z coordinates, along with the corresponding label for the point, in four columns in a formatted text file as the input. This required some manual processing, including dropping points with the four least frequent labels (essentially unusable data). Dropping four labels from the 44-label Oakland dataset also allowed us

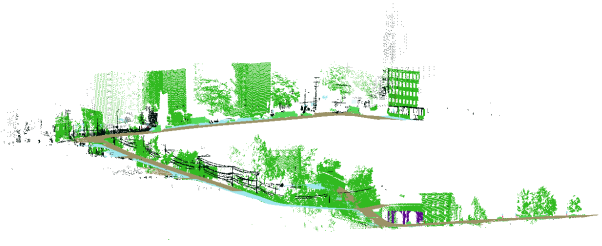

Fig. 3. A Visualization of the Complete, Preprocessed CMU Oakland Point Cloud Dataset, with the Lowest Frequency Labels Dropped from the Set.

to keep the model consistent with the total data labels in the ModelNet40 set. This would allow us to cross-test graphs made with one dataset on another dataset. A maximum number of point clouds per data file is set as well. These data points were later split into test and train datasets using a random distribution.

Once the data is read in and sorted, the creation process can begin. A label is randomly selected on a uniform distribution, and the required number of points from the corresponding data is picked to form a point cloud. In order to overcome the problem where a labeled object has too few points to create a substantially-sized point cloud for training, we add noise, or jitter, to the ground truth points, in order to create new points for the object.

The jitter fulfilled two purposes at once. First, since these points are only jittered with a value between 0.25 and 0.5 in Cartesian space, the created points would "fill in the blanks" in a sparsely-sampled object while remaining close enough in physical space to the object in question, as illustrated in Fig. 4. Secondly, the addition of random noise, and the randomized nature of label selection, allows every created dataset to be different. This would allow us to test the robustness of the model to small perturbations as well. Finally, the datasets were split into test and training datasets, of which five datasets were used for training, and two for testing.

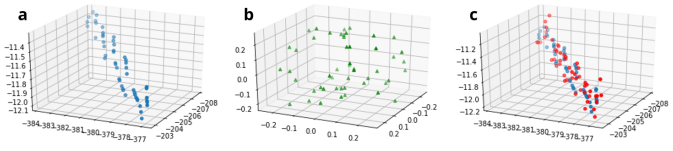

Fig. 4. A Visualization of the Jittering Process: (a) the Set of Points from the Point Cloud for the Label 'curb'; (b) Randomly Generated Jitter Values Corresponding to Each of the Points; (c) the Original Points (blue) Along with the Jittered Original Points (red) are Added to the Point Cloud

VI. Results

The modified model was run using Tensorflow with GPU acceleration. The learning rate was set to between $12 and 24 \times 10^4$, and was halved every 20 epochs or so.Our first experiment simply involved training the PointGCN model on the Oakland dataset and the ModelNet40 dataset separately, and comparing the results. With the ModelNet40, after 260 epochs of testing and training, we received an average test accuracy of 86% among the 40 input classes. For the Oakland set, four of the least frequent data labels were dropped, to ensure consistency between the total number of PointGCN outputs. After 260 epochs of testing and training, we received an average test accuracy of 91%.

For the smaller point cloud tests, sets of 64 and 128 points per point cloud were used, with only 1000 training and test examples per set (5000 training examples and 2000 test examples in total). The maximum number of epochs were set at a 100. The graph batches used by the network were first built using the Oakland dataset, and then cross-tested with the ModelNet40 dataset, in order to test the flexibility of the nodes created by the model. The final results after a 100 epochs can be seen in Table I and Table II below.

TABLE I. Final Training and Test Accuracy of CMU Oakland and Princeton ModelNet40 Datasets with a Reduced Number of Points Per Cloud. With the Base Graph for the GCN Model Built using the Oakland Dataset

	64 Points per Cloud		128 Points per Cloud	
Accuracy (%)	Oakland	ModelNet40	Oakland	ModelNet40
Training Data	95.82	90.31	96.06	90.37
Test Data	91.30	87.67	92.00	88.80
Overall	**93.56**	**88.99**	**94.03**	**89.59**

TABLE II. Final Training and Test Accuracy of CMU Oakland and Princeton ModelNet40 Datasets with a Reduced Number of Points Per Cloud, with the Base Graph for the GCN Model Built using the ModelNet40 Dataset

	64 Points per Cloud		128 Points per Cloud	
Accuracy (%)	Oakland	ModelNet40	Oakland	ModelNet40
Training Data	94.65	87.72	94.23	93.73
Test Data	83.66	84.59	85.18	87.45
Overall	**89.15**	**86.15**	**89.71**	**90.59**

The results were quite surprising. Even with a significantly reduced number of points per point cloud, both the test and training accuracies of the model on both input datasets were extremely high. When a dataset is used to construct the graph batch used by the GCN model, it demonstrates a significantly higher training and test set accuracy. Nonetheless, given that both ModelNet40 and Oakland achieve 90% accuracy and 89% accuracy respectively, on graphs not even constructed from their data, shows the flexibility and robust nature of the model. Furthermore, the runtime is also drastically decreased, especially on a GPU setup. For 7000 input point clouds with 64 points per cloud, the graph build runtime was a mere 8 minutes, and the runtime for a 100 epochs was approximately 9 minutes and 40 seconds. For the same number of input clouds at 128 points per cloud, the graph build runtime was around 13 minutes and 38 seconds, with the runtime for a 100 epochs at 12 minutes and 34 seconds. This is quite remarkable given that the runtime with over 14,000 input clouds at 2048 points per cloud has a total runtime of over a day for similar levels of accuracy, even when GPU-accelerated.

VII. Discussion

From the results of the GCN's prediction on the ModelNet40 and Oakland datasets, it is clear that the model has been extended to run with more than a single, specialized dataset. An approach using a GCN offers an adaptability that a CNN would not offer if trained on a single dataset. This

would reduce the need to have to train a model on a different dataset for a different use case every time. From cross-testing one dataset on a graph built from another dataset, it is clear that the GCN has the ability to learn and refine graphs such that it can reflect the overall structure of an input point cloud. It is also a model that has the potential to be robust to changes in the input dataset. However, on comparison with a variety of other models on the ModelNet leaderboard at [16], this model falls some ways short of the state-of-the-art methods. Nonetheless, it is still better than a lot of other models on the list and so is an approach that shows promise.

TABLE III. COMPARISON WITH STATE-OF-ART METHODS ON MODELNET40 & OAKLAND DATA SET

Method	ModelNet40	Oakland
MV3D [17]	62.94	57.31
AVOD [18]	77.90	62.03
PointPillars [19]	78.39	67.39
PointRCNN [20]	85.94	73.210
RGCN [13]	83.42	74.05
PointRGCN [13]	85.97	76.73
Extended PointRGCN(Ours)	90.59	89.71

We implemented the MV3D, AVOD, PointPillars, PointRCNN, RGCN, PointRGCN, and compared with the proposed approach. All the results reported above are the average results of 10 trials for each dataset for our proposed method taking 128 points per cloud and the results are presented in Table III. The high accuracy achieved by the model on the reduced-size point clouds was also a very curious result. As such, we looked closer at how the nature of the input data to the GCN could affect its performance. The GCN model we adapted could use one of two methods for sampling from the larger point cloud set - immediate sampling or uniform sampling. Immediate sampling just takes the first 64 or 128 points in the array from the entire point cloud. Uniform sampling attempts to take 64 or 128 samples from a uniform random probability distribution across the point cloud. If the points are unsorted, both sampling methods are equivalent. However, if the points are sorted before they are passed in, using immediate sampling shows very different effects. We attempted to sort the Oakland dataset point clouds before passing them into the GCN. We found that the accuracy on the Oakland set itself dropped to a meager 33% on the training set and 24% on the test set, while the ModelNet data achieved a 41% on the training set and 37% on the test set.

This poor performance could be attributed to the GCN receiving a poor understanding of the general structure of the point cloud from the input data. In other words, with the data sorted and sampled from the front, the GCN can essentially "visualize" only a very small fraction of the entire cloud. This does not allow it to truly infer any larger-scale features, and any features it were to infer would be grossly incorrect, given that it only has access to a small physical portion of the cloud. When the data was left unsorted, it allows the GCN to sample parts of the point cloud stochastically, thus giving it enough points to infer larger features in the cloud and encode it in the graph. We can conclude from this smaller investigation that allowing uniform, randomized sampling of the entire point cloud, even with a small number of points, can offer the same levels of accuracy while drastically decreasing graph building and prediction runtime.

VIII. CONCLUSION AND FUTURE WORK

In this paper a novel approach for extending and generalizing a GCN model to work on different point clouds such as objects in urban environments or the interiors and exterior of buildings, without suffering a significant performance loss. Further, we worked on Optimizing and generalizing current GCN model such that it can run on small computing processors such as arm cortex A7. Through our experimentation on a modified version of PointGCN, we conclude that it is possible to generalize a graph convolutional network across datasets. The proposed model seems to be able to learn and preserve underlying patterns that appear in point clouds regardless of the object represented by the point cloud, and thus make for powerful processors of the data type. Additionally, our model do not appear to lose accuracy significantly when the number of points in the point cloud input is reduced, as long as the points remain representative of the larger structure of the object at hand.

In future, we would like to extend our work by looking at how exactly the graph generated by the GCN seems to preserve the global statistics of features, and how it can effectively exploit those same statistics across datasets. We can improve the performance of our current model by adding deeper layers, where they can compute the interrelation between points to generate a feature map across the datasets. As part of future work, we also plan to see if we could combine LiDAR information with RGB images for better recognition.

REFERENCES

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[2] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[3] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *arXiv preprint arXiv:1509.09292*, 2015.

[4] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in *Proceedings of the ieee conference on computer vision and pattern recognition*, 2016, pp. 5308–5317.

[5] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, 2016, pp. 3844–3852.

[6] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[7] B. Ravi Kiran, L. Roldao, B. Irastorza, R. Verastegui, S. Suss, S. Yogamani, V. Talpaert, A. Lepoutre, and G. Trehard, "Real-time dynamic object detection for autonomous driving using prior 3d-maps," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.

[8] J. Zarzar, S. Giancola, and B. Ghanem, "Pointrgcn: Graph convolutional networks for 3d vehicles detection refinement," *arXiv preprint arXiv:1911.12236*, 2019.

[9] Y. Zhang and M. Rabbat, "A graph-cnn for 3d point cloud classification," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 6279–6283.

[10] Y. Guo, "Deep learning for 3d point clouds: A survey," *https://arxiv.org/abs/1912.12033*.

[11] P. V. D. I. Shuman and P. Frossard, "Chebyshev polynomial approximation for distributed signal processing," *n Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems 2011*, 2011.

[12] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," *arXiv preprint arXiv:1703.06103*, 2017.

[13] Y. Lingfan, L. Mufei, and Z. Zheng, "Relational graph convolutional network," *DGL*, 2018.

[14] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[15] N. V. D. Munoz, J. A. Bagnell and M. Hebert, "Contextual classification with functional max-margin markov networks," *In IEEE Conference on Computer Vision and Pattern Recognition 2009*, 2009.

[16] Princeton, "Princeton modelnet." [Online]. Available: https://modelnet.cs.princeton.edu/

[17] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, in *Multi-view 3d object detection network for autonomous driving*. IEEE, 2017, pp. 1907–1915.

[18] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, in *Point 3d proposal generation and object detection from view aggregation*. IEEE, 2018, pp. 1–8.

[19] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

[20] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 770–779.