Analysis of Dynamic Graph CNN for Learning on Point Clouds

Matthieu Mérigot-Lombard
Ecole des Ponts ParisTech
Paris, France
matthieu.merigot-lombard@eleves.enpc.fr

Guillaume Vindry
Ecole des Mines de Paris
Paris, France
guillaume.vindry@etu.minesparis.psl.eu

ABSTRACT

In this report, we introduce the Dynamic Graph CNN (DGCNN) architecture [6], presenting the context and reproducing the main experiments. We review influential prior works and highlight the novel insights introduced by the authors, and we reimplement the model in PyTorch for both classification and segmentation tasks, successfully reproducing the results using pretrained weights. Additionally, we recreate figures illustrating how the dynamic recalculation of graphs at each step allows for non-local interactions between points, and demonstrating the model's robustness to homogeneous variations in point density. Finally, we discuss the limitations of this architecture and propose potential solutions.

1 INTRODUCTION

Point clouds are essential in various 3D vision tasks, including object recognition, object or scene segmentation. Use cases include self-driving vehicles, autonomous robots and medical imaging. Unlike traditional 2D images, point cloud data is by nature irregular and unordered, posing significant challenges for conventional processing techniques. The DGCNN aims to provide an innovative approach that leverages both dynamic graph convolutional operations and local geometry information to effectively learn and capture the relationships among points in a point cloud.

This report first makes a review of the DGCNN methodology, highlighting its novel contributions to the field of point cloud processing. We then try to reproduce the experiments conducted by the original authors to validate their results and critically evaluate the design choices made in the development of the model. Furthermore, we will discuss the practical applications of the pretrained DGCNN model, its limitations, and potential areas for future research and enhancement. Through this comprehensive analysis, we aim to offer valuable insights into the strengths and weaknesses of the DGCNN approach.

2 BACKGROUND AND PREVIOUS METHODS

In this report, we will focus on the tasks of object classification and object segmentation of point clouds.

2.1 Geometric feature engineering

Early approaches relied on manually engineering intrinsic and extrinsic geometric features within the point cloud, which were then used as input for trainable models. Another strategy involved using convolutions similar to those in CNNs . While CNNs are highly effective for image classification and segmentation, defining convolutions on point clouds is challenging due to the absence of an underlying grid that encodes proximity information. One solution among many is to create a voxel-like structure within the point cloud to form a grid [3], however these methods require extensive preprocessing of the data, which can be computationally intensive and potentially lead to information loss between the raw data and the model.

2.2 PointNet

The PointNet model [1] directly takes the raw point cloud as input, bypassing intermediate preprocessing steps. This approach allows a more canonical representation of the point cloud data, facilitating the computation of other representations (e.g. mesh, depth map).

The architecture of the PointNet model is as follows. The input features are the 3D coordinates of each point. A transform layer applies a data-dependent 3D transformation to align the point cloud. Each point is then individually transformed by a shared MLP, followed by another transform layer that aligns the cloud in the new embedding space. The data is processed by another MLP and then max-pooled over all points to form global features for the entire cloud. A final MLP outputs the prediction. The pooling layer ensures the model is permutation-invariant which is crucial because of the unordered nature of point clouds.

One limitation of this approach is its difficulty in learning local features, as the only interaction between points occurs in the max pooling layer (and the data-dependent transform, which is primarily for alignment). The geometric information is solely derived from the 3D coordinates, which is insufficient. To address this, an improved architecture called PointNet++ was developed [4]. PointNet++ employs a hierarchical neural network architecture: smaller PointNet models are applied to neighborhoods of points, and the results are pooled. This operation is repeated with increasing granularity in subsequent layers, allowing for multi-scale feature processing.

2.3 CNNs on graphs

Since the DGCNN model aims to generalize CNNs to point clouds using a graph of proximity, we will present one of most common methods to define convolution on graphs, message passing, introduced in [2]. It consists in gathering at each node a function of each of its neighbors in the graph.

At the t-th message passing, at node v:

$$m_v^{t+1} = \sum_{w \in \mathcal{N}(v)} M_t\left(h_v^t, h_w^t, e_{vw}\right)$$
$$h_v^{t+1} = U_t\left(h_v^t, m_v^{t+1}\right)$$

With $\mathcal{N}(v)$ the neighborhood of v, M_t the "message" function, and U_t a function to update the nodes. This framework makes it possible to create various types of convolutions depending on the usage. These operations have similar properties to convolutions on images.

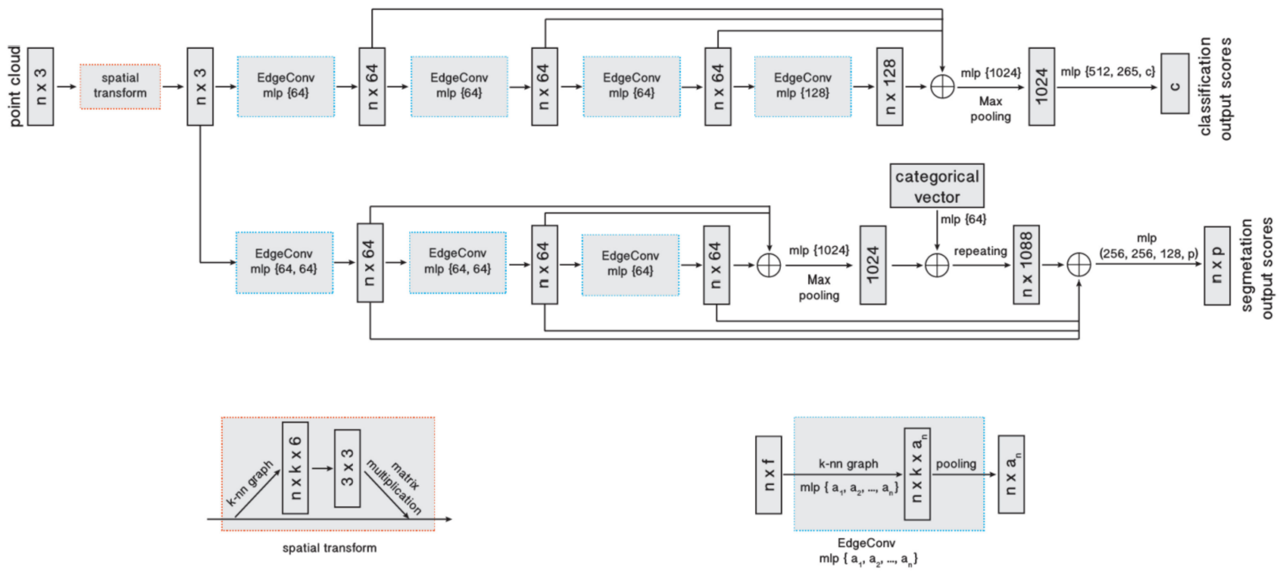

Figure 1: Architecture of the object classification (top branch) and the object segmentation (bottom branch) presented in the original DGCNN paper.

3 THEORETICAL BASIS

The authors aimed to develop an analog of CNNs for point clouds. While this concept had been explored in previous literature, their key innovation was to draw inspiration from PointNet by using only the raw point cloud as input, thereby learning the "grid" during training.

The "grid" for convolution is represented as a directed graph, where each point is connected to its k nearest neighbors. Due to the asymmetry of the k-NN relationship, this graph is directed. A convolution-like operation, called EdgeConv, is applied to this graph to update the node features. EdgeConv is similar to message passing in graph neural networks [2], where node features are iteratively updated based on their neighbors.

By recomputing the graph at each layer, the model can dynamically adjust the neighborhood relationships between points as feature representations become more refined in subsequent layers. This allows the model to capture more abstract geometric structures, ensuring that points with similar high-level features are brought closer together in the feature space.

3.1 Feature transform

Similar to the PointNet model, the point cloud is initially aligned using a 3D transformation represented by a 3x3 matrix. This alignment is crucial because it standardizes the orientation and position of the point cloud, making it easier for the model to learn important features. The transformation must depend on the input data to be effective, so a small neural network, known as the T-Net, is employed to model this dependency. The T-Net learns to predict the optimal transformation matrix for each input point cloud, making sure that the points are consistently aligned before further processing.

3.2 Edge convolution

The authors developed a general framework for dynamic graph processing using an arbitrary EdgeConv function. Its general shape at node i with features x_i and neighbors $\mathcal{N}_i$ is:

$$\text{EdgeConv}(\mathbf{x_i}) = \square_{x_j \in \mathcal{N}_i \cup \{x_i\}} h_\Theta(x_i, x_j)$$

The operator $\square$ is a symmetric aggregator like a sum, a mean or a maximum.

PointNet's fully connected layers can be interpreted within this framework as the absence of a graph in PointNet is equivalent to the following: for each point i, the neighborhood $\mathcal{N}_i$ is empty, i.e. the only edge feature is from the self-loop, and the function $h_\Theta(x_i, x_i)$ is $\text{MLP}_\Theta(x_i)$. Thus, the edge convolution operation in PointNet can be expressed as:

$$\text{EdgeConv}_\Theta^{\text{PointNet}}(x_i) = \text{MLP}_\Theta(x_i)$$

However, this approach does not consider the local geometry of the point cloud. An alternative would be to use a function that depends solely on the distances to each neighbor. This representation would encode "patches" of the point cloud without relying on an absolute frame of reference, making the function invariant to translations in the feature space.

After evaluating various options, the authors determined that the most effective edge convolution operation for their use case incorporates both the features of x_i and the distances to x_j. This approach allows the model to capture more detailed and contextually relevant geometric information. The two terms are then combined linearly with learned weights, and the result is passed through a ReLU activation function to introduce nonlinearity such that:

$$h_\Theta(x_i, x_j) = \text{ReLU}(\theta \cdot (x_j - x_i) + \phi \cdot x_i)$$

Using the maximum as the aggregation function, the edge convolution operation in DGCNN can be expressed as:

$$\text{EdgeConv}_{\Theta}^{\text{DGCNN}}(x_i) = \max_{i} \text{ReLU}(\theta \cdot (x_j - x_i) + \phi \cdot x_i)$$

3.3 Model architecture

The authors detail two types of architectures. First, the top branch of Figure 1 illustrates the architecture of the object classification network. The input point cloud is aligned using T-Net, followed by a series of EdgeConv layers that extract local geometric features. Features from each layer are pooled to form a global feature vector, which is then passed through a MLP to produce the final classification output.

The bottom branch of Figure 1 is a segmentation network which has a similar initial structure, starting with T-Net for alignment and EdgeConv layers for feature extraction. However, instead of only pooling, it combines the features with an additional categorical vector. This categorical vector is a one-hot encoded representation of the object's class, such as those in the ShapeNet dataset which includes 16 classes: airplane, bag, cap, car, chair, earphone, guitar, knife, lamp, laptop, motor, mug, pistol, rocket, skateboard and table. The combined features are passed through an MLP which outputs per-point segmentation scores.

4 EVALUATION

We will now attempt to replicate the results of the DGCNN paper, demonstrating how it leverages geometric information to predict the class of an object or its segmentation.

The code used to generate the following results is available at https://github.com/matthieuml/DGCNN. The results were obtained using inference on Linux 6.12.1, Python 3.12.7, and Torch 2.5.1, running on an NVIDIA GTX 1050Ti Mobile (additional package details can be found in the GitHub repository).

	Mean Class Acc.	Total Acc.	mIoU
PointNet (Qi et al. 2017)	86.0	89.2	-
PointNet++ (Qi et al. 2017)	-	90.7	-
DGCNN (k=10, n=1024)	81.4	86.3	71.5
DGCNN (k=20, n=1024)	90.1	93.0	84.1
DGCNN (k=40, n=1024)	80.9	85.1	71.5
DGCNN (k=60, n=1024)	61.4	67.3	49.2
DGCNN (k=10, n=2048)	54.9	62.1	42.8
DGCNN (k=20, n=2048)	87.6	90.4	80.0
DGCNN (k=40, n=2048)	89.5	92.7	83.1
DGCNN (k=60, n=2048)	87.3	90.3	80.0

Table 1: Comparison of mean class accuracy, total accuracy, and the mean intersection over union for our implementation of the DGCNN model against the PointNet and PointNet++ models on the test partition of ModelNet40 dataset. Results are shown for different values of k (10, 20, 40, 60) and different point cloud sizes n (1024, 2048). The values for the PointNet and PointNet++ results are taken from the DGCNN paper.

4.1 Object Classification

To reproduce the results, we will use the PyTorch implementation of the DGCNN classification model provided by the authors on their GitHub repository https://github.com/WangYueFt/dgcnn, as it contains the pretrained model weights. It is important to note that the architecture in this repository differs slightly from the one shown in Figure 1. Specifically, it does not include a T-Net for point cloud alignment, and the max pooling layer is replaced by a concatenation of a pooling layer and an average pooling layer. We used the ModelNet40 test dataset to evaluate the model's performance.

Table 1 presents the mean class accuracy, overall accuracy, and mean intersection over union of the classification model for different hyperparameters. The results are not identical to those in the paper, as the paper used the TensorFlow implementation with the original architecture. However, the results are very close and exhibit similar behavior with changes in hyperparameters.

First, it shows that with the right choice of hyperparameters, the DGCNN model can indeed outperform both the PointNet and PointNet++ models, as promised by the original paper. Additionally, we observe that for a fixed number of points in the point cloud, the different metrics increase with the number of neighbors up to a point, and then decrease. The optimal value of k appears to be higher as the number of points increases: $k = 20$ is optimal for 1024 points, while $k = 40$ is optimal for 2048 points.

In Figure 2, we plotted the structure of the k-NN for various point clouds with 2048 points from the ModelNet40 dataset at each EdgeConv. This was done to understand how the inherent geometry generated by the features evolves. The results indicate that the model captures more semantic distance as the layers deepen. Notable examples include the top of the table, where all points on the top are considered close to one in the edge, and the clearly visible edge of the couch. Thus, with the appropriate choice of k, incorporating EdgeConv in the architecture enables the model to better capture geometric information, improving prediction accuracy.

To investigate the robustness of the model to the homogeneous variation of the density of points in the cloud, we plot the accuracy of the model as the number of points is uniformly reduced in Figure 5. Starting with 2048 points from the ModelNet40 dataset, the accuracy significantly declines below a proportion of 0.7. Figure 6 illustrates the visual effect of uniform density reduction.

4.2 Object Segmentation

The second architecture of the DGCNN model is designed for object segmentation. This model takes two inputs: the point cloud and a categorical vector indicating the class to which the point cloud belongs. The number of segmentations for each object class is predefined, determining the shape of the output. To test this model architecture, we use the pretrained weights available from another GitHub repository https://github.com/antao97/dgcnn.pytorch which replicated the experiments using PyTorch. The main challenge in this section was obtaining the ShapeNet dataset used in the paper. In the end, we found only a raw copy of the dataset that was

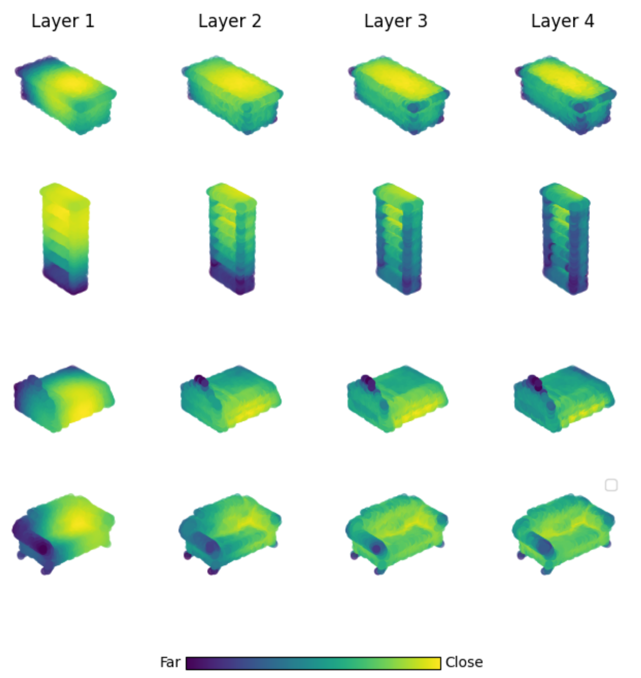

Figure 2: Structure of the k-NN at each EdgeConv in the classification model for various point clouds, each with 2048 points, mapped to 40 object classes from the ModelNet40 dataset. The distances between one point and the others in the point cloud are shown. Note how similar structures appear closer together as the model goes deeper, such as the back of the shelves, the border of the bed, and the edges of the couch.

not split into train and test sets. Therefore, we focused on performing some inferences with the model to evaluate its performance.

Figure 3 presents the results of the segmentation prediction on several point clouds with 2048 points from the ShapeNet dataset. Overall, the results are very impressive, with the segmentations being nearly identical to the ground truth. Only minor differences are observed, such as on the wheels of the car and the top of the guitar. In the case of a more complex point cloud such as the top plane in Figure 4, the model mistakenly identifies the cylindrical shapes of the wings as reactors. This is not surprising, as even humans might have difficulty recognizing them as wings. Thus, we conclude that the model is working as expected.

5 LIMITATIONS AND EXTENSIONS

The segmentation part of the DGCNN model has a fixed number of segmentation classes for each object class. Since objects within the same class can vary significantly in complexity and structure, as illustrated in Figure 4, a fixed number of segmentation classes may not adequately capture the nuances of more complex objects, resulting in suboptimal segmentation. Therefore, if new point clouds with a different segmentation class are added, the model would likely need to be retrained entirely or at least the last layers.

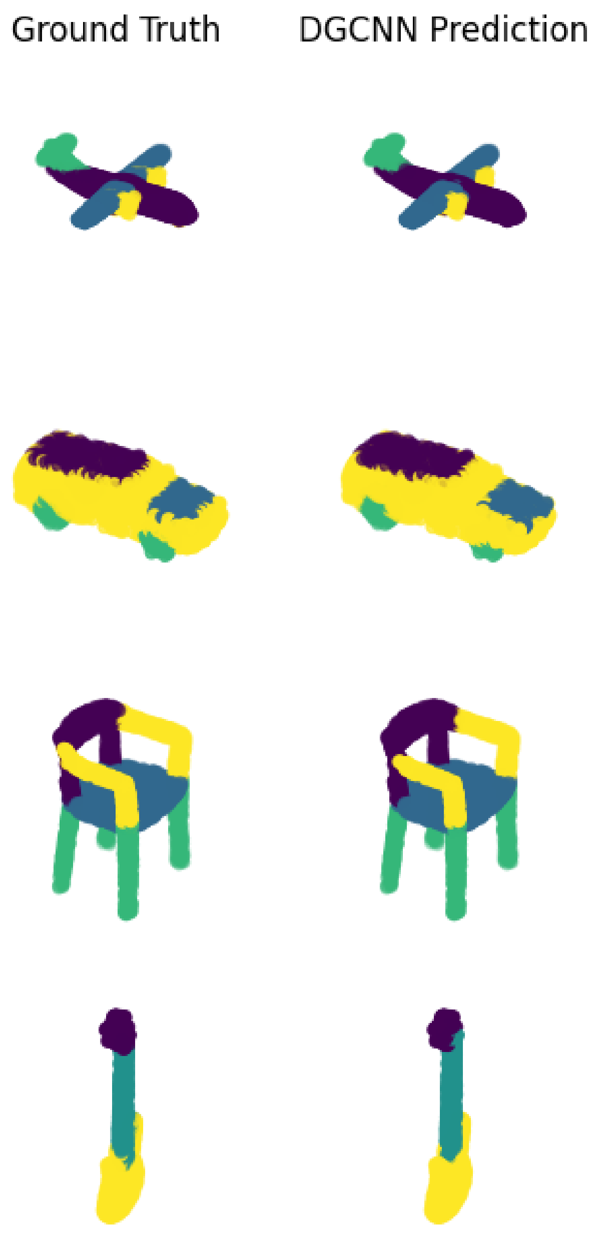

Figure 3: Segmentation prediction from the segmentation model for various point clouds with 2048 points across 16 classes from the ShapeNet dataset.

Another limitation of this architecture is that the k-NN approach can be ineffective on point clouds with inhomogeneous density. In regions with high point density, the k-NN may not encompass enough points to cover a sufficiently large surface for capturing geometric features. Since the input feature space is not restricted

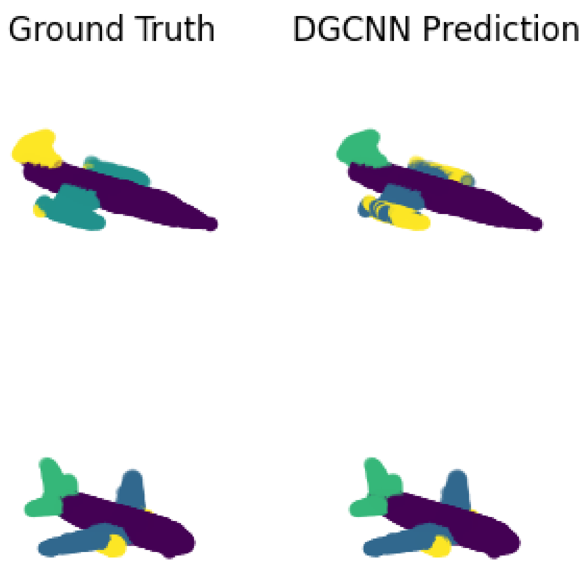

Figure 4: Segmentation prediction from the segmentation model for various point clouds with 2048 points, shown for two different planes from the ShapeNet dataset. Note that the first plane has only three segmented parts in the ground truth, while the model predicts four segments.

to 3D, we could incorporate additional geometric information as input to account for local features, regardless of point density. An approach proposed by [5] suggests including more local geometric features as input and improving graph computation by considering geometry instead of relying solely on k-NN, thereby addressing the density issues.

A distinct method to introduce non-local interactions in point clouds is by adapting the Transformer architecture, particularly its self-attention mechanism, as demonstrated in the Point-BERT paper [7]. This approach involves creating local point patches that are embedded into a feature space using a variational autoencoder. During training, some patches are randomly masked, and the self-attention mechanism is applied to the remaining patches. The model is trained to predict the masked patches, allowing it to capture both local geometric features within patches and global relationships between them.

Finally, the k-NN calculation is computationally intensive and difficult to parallelize. Although the authors claim that their method is faster than PointNet, our benchmarks indicate that PointNet is approximately 50 times faster at inference. The authors' experiments show that dynamic graph computation improves classification accuracy by less than 1%, raising questions about its practical usefulness beyond benchmark scenarios. Furthermore, traditional CNNs on 2D images do not incorporate this type of non-locality.

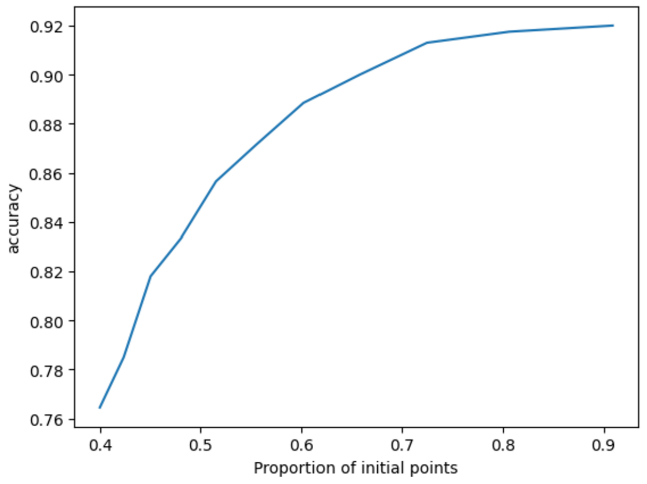

Figure 5: Robustness of the classification model to reducing the point density on the ModelNet40 dataset starting from 2048 points.

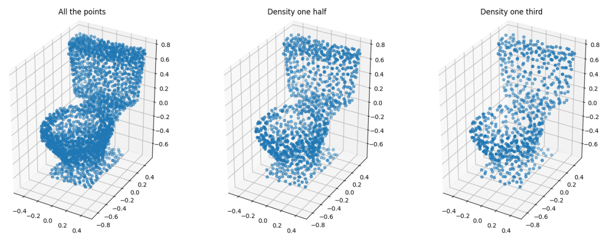

Figure 6: Reduction in uniform density (2048, 1024, and 683 points) for robustness experiments on a point cloud from ModelNet40.

6 CONCLUSION

The DGCNN model is very effective for classification and segmentation of raw point clouds, allowing direct processing of data from 3D scanners. It has demonstrated improved performance compared to previous state-of-the-art models such as PointNet and PointNet++. However, the dynamic graph approach is difficult to parallelize and can be computationally expensive. In our experiments, it is a lot slower than PointNet at inference. It also faces challenges with inhomogeneous point density. Therefore, new approaches, such as [5], have been proposed to address these limitations.

REFERENCES

[1] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation . In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 77–85. https://doi.org/10.1109/CVPR.2017.16

[2] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. *CoRR* abs/1704.01212 (2017). arXiv:1704.01212 http://arxiv.org/abs/1704.01212

[3] Roman Klokov and Victor Lempitsky. 2017. Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 863–872. https://doi.org/10.1109/ICCV.2017.99

[4] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *CoRR* abs/1706.02413 (2017). arXiv:1706.02413 http://arxiv.org/abs/1706.02413

[5] Siddharth Srivastava and Gaurav Sharma. 2021. Exploiting Local Geometry for Feature and Graph Construction for Better 3D Point Cloud Processing with Graph Neural Networks. *CoRR* abs/2103.15226 (2021). arXiv:2103.15226 https://arxiv.org/abs/2103.15226

[6] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. Graph.* 38, 5, Article 146 (Oct. 2019), 12 pages. https://doi.org/10.1145/3326362

[7] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. 2021. Point-BERT: Pre-training 3D Point Cloud Transformers with Masked Point Modeling. *CoRR* abs/2111.14819 (2021). arXiv:2111.14819 https://arxiv.org/abs/2111.14819