Grid-GCN for Fast and Scalable Point Cloud Learning

Qiangeng Xu[1]   Xudong Sun[2]   Cho-Ying Wu[1]   Panqu Wang[2]   Ulrich Neumann[1]

[1]University of Southern California   [2]Tusimple, Inc

{qiangenx,choyingw,uneumann}@usc.edu   {xudong.sun,panqu.wang}@tusimple.ai

Abstract

Due to the sparsity and irregularity of the point cloud data, methods that directly consume points have become popular. Among all point-based models, graph convolutional networks (GCN) lead to notable performance by fully preserving the data granularity and exploiting point inter-relation. However, point-based networks spend a significant amount of time on data structuring (e.g., Farthest Point Sampling (FPS) and neighbor points querying), which limit the speed and scalability. In this paper, we present a method, named Grid-GCN, for fast and scalable point cloud learning. Grid-GCN uses a novel data structuring strategy, Coverage-Aware Grid Query (CAGQ). By leveraging the efficiency of grid space, CAGQ improves spatial coverage while reducing the theoretical time complexity. Compared with popular sampling methods such as Farthest Point Sampling (FPS) and Ball Query, CAGQ achieves up to $50\times$ speed-up. With a Grid Context Aggregation (GCA) module, Grid-GCN achieves state-of-the-art performance on major point cloud classification and segmentation benchmarks with significantly faster runtime than previous studies. Remarkably, Grid-GCN achieves the inference speed of 50fps on ScanNet using 81920 points as input. The supplementary [1] and the code [2] are released.

1. Introduction

Point cloud data is popular in applications such as autonomous driving, robotics, and unmanned aerial vehicles. Currently, LiDAR sensors can generate millions of points a second, providing dense real-time representations of the world. Many approaches are used for point cloud data processing. *Volumetric models* are a family of models that transfer point cloud to spatially quantized voxel grids and use a volumetric convolution to perform computation in the grid space [28, 45, 28]. Using grids as data structuring methods, volumetric approaches associate points to locations in grids, and 3D convolutional kernels gather information from neighboring voxels. Although grid data struc-
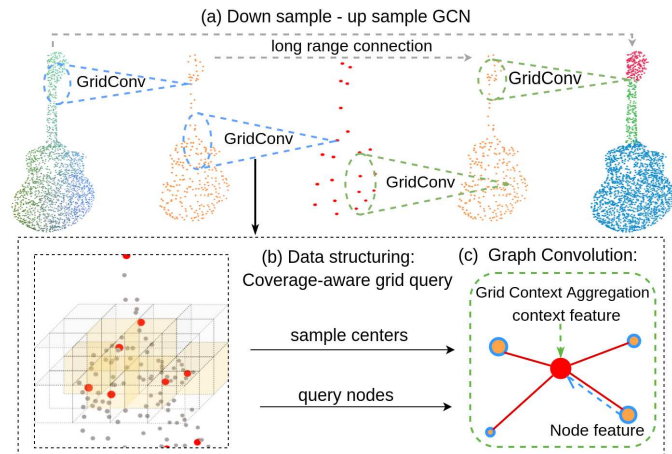
Figure 1: Overview of the Grid-GCN model. (a) Illustration of the network architecture for point cloud segmentation. Our model consists of several GridConv layers, and each can be used in either a downsampling or an upsampling process. A GridConv layer includes two stages: (b) For the data structuring stage, a Coverage-Aware Grid Query (CAGQ) module achieves efficient data structuring and provides point groups for efficient computation. (c) For the convolution stage, a Grid Context Aggregation (GCA) module conducts graph convolution on the point groups by aggregating local context.

tures are efficient, high voxel resolution is required to preserve the granularity of the data location. Since computation and memory usage grows cubically with the voxel resolution, it is costly to process large point clouds. In addition, since approximately 90% of the voxels are empty for most point clouds[51], significant computation power may be consumed by processing no information.

Another family of models for point cloud data processing is *Point-based models*. In contrast to volumetric models, point-based models enable efficient computation but suffer from inefficient data structuring. For example, Point-Net [29] consumes the point cloud directly without quantization and aggregates the information at the last stage of the network, so the accurate data locations are intact but the computation cost grows linearly with the number of points. Later studies [30, 47, 41, 37, 46] apply a downsampling strategy at each layer to aggregate information

into point group centers, therefore extracting fewer representative points layer by layer (Figure 1(a)). More recently, graph convolutional networks (GCN) [32, 39, 21, 49] are proposed to build a local graph for each point group in the network layer, which can be seen as an extension of the PointNet++ architecture [30]. However, this architecture incurs high data structuring cost (e.g., FPS and k-NN). Liu et al. [27] show that the data structuring cost in three popular point-based models [23, 47, 41] is up to 88% of the overall computational cost. In this paper, we also examine this issue by showing the trends of data structuring overhead in terms of scalability.

This paper introduces Grid-GCN, which blends the advantages of volumetric models and point-based models, to achieve efficient data structuring and efficient computation at the same time. As illustrated in Figure 1, our model consists of several *GridConv* layers to process the point data. Each layer includes two stages: a data structuring stage that samples the representative centers and queries neighboring points; a convolution stage that builds a local graph on each point group and aggregates the information to the center.

To achieve efficient data structuring, we design a **Coverage-Aware Grid Query (CAGQ)** module, which 1) accelerates the center sampling and neighbor querying, and 2) provides more complete coverage of the point cloud to the learning process. The data structuring efficiency is achieved through voxelization, and the computational efficiency is obtained through performing computation only on occupied areas. We demonstrate CAGQ's outstanding speed and space coverage in Section 4.

To exploit the point relationships, we also describe a novel graph convolution module, named **Grid Context Aggregation (GCA)**. The module performs *Grid context pooling* to extract context features of the grid neighborhood, which benefits the edge relation computation without adding extra overhead.

We demonstrate the Grid-GCN model on two tasks: point cloud classification and segmentation. Specifically, we perform the classification task on the ModelNet40 and ModelNet10 [43], and achieve the state-of-the-art overall accuracy of 93.1% (no voting), while being on average 5× faster than other models. We also perform the segmentation tasks on ScanNet [8] and S3DIS [1] dataset, and achieve 10× speed-up on average than other models. Notably, our model demonstrates its ability on real-time large-scale point-based learning by processing 81920 points in a scene within 20 ms. (see Section 5.3.1).

2. Related Work

Voxel-based methods for 3D learning To extend the success of convolutional neural network models[12, 13] on 2D images, Voxnet and its variants [28, 43, 38, 4, 6] start to transfer point cloud or depth map to occupancy grid and apply volumetric convolution. To address the problem of cubically increased memory usage, OctNet[31] constructs tree structures for occupied voxels to avoid the computation in the empty space. Although efficient in data structuring, the drawback of the volumetric approach is the low computational efficiency and the loss of data granularity.

Point-based methods for point cloud learning Point-based models are first proposed by [29, 30], which pursues the permutation invariant by using pooling to aggregate the point features. Approaches such as kernel correlation [2, 42] and extended convolutions [36] are proposed to better capture local features. To solve the ordering ambiguity, PointCNN [23] predicts the local point order, and RSNet [14] sequentially consumes points from different directions. The computation cost in point-based methods grows linearly with the number of input points. However, the cost of data structuring has become the performance bottleneck on large-scale point clouds.

Data structuring strategies for point data Most point-based methods [30, 23, 37, 26] use FPS [10] to sample evenly spread group centers. FPS picks the point that maximizes the distance to the selected points. If the number of centers is not very small, the method takes $O(N^2)$ computation. An approximate algorithm [9] can be $O(NlogN)$. Random Point Sampling (RPS) has the smallest possible overhead, but it's sensitive to density imbalance. Our CAGQ module has the same complexity as RPS, but it performs the sampling and neighbors querying in one shot, which is even faster than RPS with Ball Query or k-NN (see Table 2). KPConv [36] uses a grid sub-sampling to pick points in occupied voxels. Unlike our CAGQ, the strategy cannot query points in the voxel neighbors. CAGQ also has a Coverage-Aware Sampling (CAS) algorithm that optimizes the center selections, which can achieve better coverage than FPS.

Alternatively, SO-Net [22] builds a self-organizing map. KDNet [15] uses kd-tree to partition the spaces. PATs[48] uses Gumble Subset Sampling to replace FPS. SPG [19] uses a clustering method to group points as super points. All of these methods are either slow in speed or need structure preprocessing. The lattice projection in SPLATNet [33, 11] preserves more point details than voxel space, but it is slower. Studies such as VoxelNet [51, 20] combines the point-based and volumetric methods by using PointNet [29] inside each voxel and applying voxel convolution. A concurrent high-speed model PVCNN [27] uses similar approaches but does not reduce the number of points in each layer progressively. Grid-GCN, yet, can down-sample a large number of points through CAGQ, and aggregate information by considering node relationships in local graphs.

GCN for point cloud learning Graph convolutional networks have been widely applied on point cloud learning [41, 18, 17]. A local graph is usually built for each point
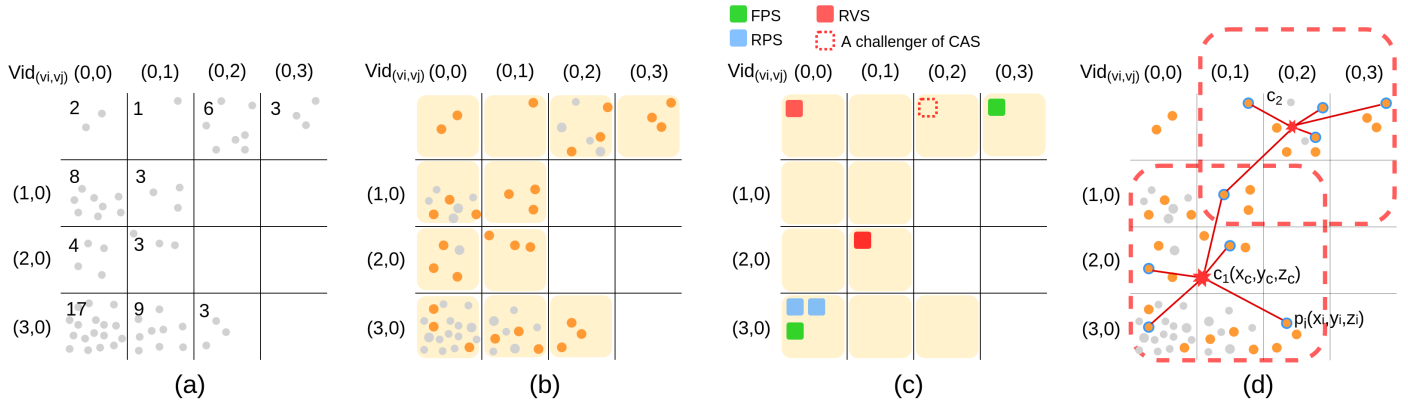
Figure 2: Illustration of Coverage-Aware Grid Query (CAGQ). Assume we want to sample $M = 2$ point groups and query $K = 5$ node points for each group. **(a)** The input is N points (grey). The voxel id and number of points is listed for each occupied voxel. **(b)** We build voxel-point index and store up to $n_v = 3$ points (yellow) in each voxel. **(c)** Comparison of different sampling methods: FPS and RPS prefer the two centers inside the marked voxels. Our RVS could randomly pick any two occupied voxels (e.g. (2,0) and (0,0)) as center voxels. If our CAS is used, voxel (0,2) will replace (0,0). **(d)** Context points of center voxel (2,1) are the yellow points in its neighborhood (we use 3×3 as an example). CAGQ queries 5 points (yellow points with blue ring) from these context points, then calculate the locations of the group centers.

group, and GCN aggregates point data according to relations between points. SpecConv[37] blends the point features by using a graph Fourier transformation. Other studies model the edge feature between centers and nodes. Among them, [47, 26, 17, 41, 49] use the geometric relations, while [6, 39] explore semantic relations between the nodes. Apart from those features, our proposed Grid Context Aggregation module considers coverage and extracts the context features to compute the semantic relation.

3. Methods

3.1. Method Overview

As shown in Figure 1, Grid-GCN is built on a set of Grid-Conv layers. Each GridConv layer processes the information of N points and maps them to M points. The downsampling GridConv ($N > M$) is repeated several times until a final feature representation is learned. This representation can be directly used for tasks such as classification or further up-sampled by the upsampling GridConv layers ($N < M$) in segmentation tasks.

GridConv consists of two modules:

1. A Coverage-Aware Grid Query (CAGQ) module that samples M point groups from N points. Each group includes K node points and a group center. In the upsampling process, CAGQ takes centers directly through long-range connections, and only queries node points for these centers.

2. A Grid Context Aggregation (GCA) module that builds a local graph for each point group and aggregates the information to the group centers. The M group centers are passed as data points for the next layer.

We list all the notations in the supplementary for clarity.

3.2. Coverage-Aware Grid Query (CAGQ)

In this subsection, we discuss the details of the CAGQ module. Given a point cloud, CAGQ aims to effectively structure the point cloud, and ease the process of center sampling and neighbor points querying. To perform CAGQ, we first voxelize the input space by setting up a voxel size (v_x, v_y, v_z). We then map each point to a voxel index $Vid(u, v, w) = floor(\frac{x}{v_x}, \frac{y}{v_y}, \frac{z}{v_z})$. Here we only store up to n_v points in each voxel.

Let O_v denote all of the non-empty voxels. We then sample M *center voxels* $O_c \subseteq O_v$. For each center voxel v_i, we define its voxel neighbors $\pi(v_i)$ as the voxels within the neighbor-hood of a center voxel. In Figure 2d, $\pi(v(2, 1))$ are the 3X3 voxels inside the red box. We call the stored points inside $\pi(v_i)$ as *context points*. Since we build the point-voxel index in the previous step, CAGQ can quickly retrieve context points for each v_i.

After that, CAGQ picks K *node points* from the context points of each v_i. We calculate the barycenter of node points in a group, as the location of the group center. This entire process is shown in Figure 2.

Two problems remain to be solved here. (1) How do we sample center voxels $O_c \subseteq O_v$. (2) How do we pick K nodes from context points in $\pi(v_i)$.

To solve the first problem, we propose our **center voxels sampling** framework, which includes two methods:

1. *Random Voxel Sampling (RVS)*: Each occupied voxel will have the same probability of being picked. The group centers calculated inside these center voxels are more evenly distributed than centers picked on input points by RPS. We discuss the details in Section 4.

2. *Coverage-Aware Sampling (CAS)*: Each selected cen-

ter voxel can cover up to λ occupied voxel neighbors. The goal of CAS is to select a set of center voxels O_c such that they can cover the most occupied space. Seeking the optimal solution to this problem requires iterating all combinations of selections. Therefore, we employ a greedy algorithm to approach the optimal solution: We first randomly pick M voxels from O_v as *incumbents*; From all of the unpicked voxels, we iteratively select one to challenge a random incumbent each time. If adding this *challenger* (and in the meantime removes the incumbent) gives us better coverage, we replace the incumbent with the challenger. For a challenger v_C and an incumbent v_I, the heuristics are calculated as:

$$\delta(x) = \begin{cases} 1, & \text{if } x = 0. \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

$$H_{add} = \sum_{V \in \pi(V_C)} \delta(C_V) - \beta \cdot \frac{C_V}{\lambda} \tag{2}$$

$$H_{rmv} = \sum_{V \in \pi(V_I)} \delta(C_V - 1) \tag{3}$$

where λ is the amount of neighbors of a voxel and C_V is the number of incumbents covering voxel V. H_{add} represents the coverage gain if adding V_C (penalize by a term of over-coverage). H_{rmv} represents the coverage loss after removing V_I. If $H_{add} > H_{rmv}$, we replace the incumbent by the challenger voxel. If we set β as 0, each replacement is guaranteed to improve the space coverage.

Comparisons of those methods are further discussed in section 4.

Node points querying CAGQ also provides two strategies to pick K node points from context points in $\pi(v_i)$.

1. *Cube Query*: We randomly select K points from context points. Compared to the Ball Query used in PointNet++ [30], Cube Query can cover more space when point density is imbalanced. In the scenario of Figure 2, Ball Query samples K points from all raw points (grey) and may never sample any node point from voxel (2,1) which only has 3 raw points.

2. *K-Nearest Neighbors*: Unlike traditional k-NN where the search space is all points, k-NN in CAGQ only need to search among the context points, making the query substantially faster (We also provide an optimized method in the supplementary materials). We will compare these methods in the next section.

3.3. Grid Context Aggregation

For each point group provided by CAGQ, we use a Grid Context Aggregation (GCA) module to aggregate features from the node points to the group center. We first construct a local graph $G(V, E)$, where V consists of the group center and K node points provided by CAGQ. We then connect each node point to the group center. GCA projects a node
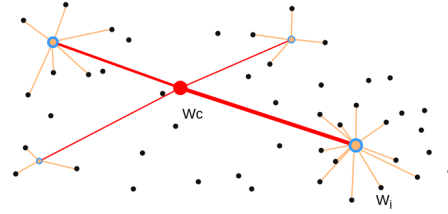

Figure 3: The red point is the group center. Yellow points are its node points. Black points are node points of the yellow points in the previous layer. The coverage weight is an important feature as it encodes the number of black points that have been aggregated to each yellow point.

point's features f_i to $\widetilde{f}_i$. Based on the edge relation between the node and the center, GCA calculates the contribution of $\widetilde{f}_i$ and aggregates all these features as the feature of the center $\widetilde{f}_c$. Formally, the GCA module can be described as

$$\widetilde{f}_{c,i} = e(\chi_i, f_i) * \mathcal{M}(f_i) \tag{4}$$

$$\widetilde{f}_c = \mathcal{A}(\{\widetilde{f}_{c,i}\}, i \in 1, ..., K) \tag{5}$$

where $\widetilde{f}_{c,i}$ is the contribution from a node, and χ_i is the xyz location of the node. $\mathcal{M}$ is a multi-layer perceptron (MLP), e is the edge attention function, and $\mathcal{A}$ is the aggregation function. The edge attention function e has been explored by many previous studies [47, 6, 39]. In this work, we design a new edge attention function with the following improvements to better fit into our network architecture (Figure 4):

Coverage Weight Previous studies [47, 26, 17, 41, 49] use χ_c of the center and χ_i of a node to model edge attention as a function of geometric relation (Figure 4b). However, the formulation ignores the underlying contribution of each node point from previous layers. Intuitively, node points with more information from previous layers should be given more attention. We illustrate this scenario in Figure 3. With that in mind, we introduce the concept of *coverage weight*, which is defined as the number of points that have been aggregated to a node in previous layers. This value can be easily computed in CAGQ, and we argue that coverage weight is an important feature in calculating edge attention (see our ablation studies in Table 6).

Grid Context Pooling Semantic relation is another important aspect when calculating the edge attention. In previous works [6, 39], semantic relation is encoded by using the group center's features f_c and a node point's features f_i, which requires the group center to be selected from node points. In CAGQ, since a group center is calculated as the barycenter of the node points, we propose *Grid context pooling* that extracts context features f_{cxt} by pooling from all context points, which sufficiently covers the entire grid space of the local graph. Grid context pooling brings the following benefits:

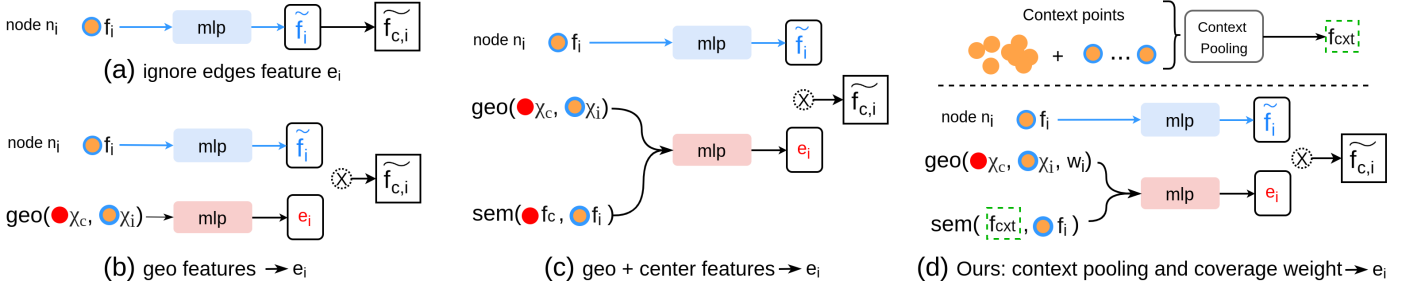- f_{cxt} models the features of a virtual group center, which

Figure 4: Different strategies to compute the contribution $\widetilde{f_{c,i}}$ from a node n_i to its center c. f_i, χ_i are the feature maps and the location of n_i. e_i is the edge feature between n_i and c calculated from the edge attention function. **(a)** Pointnet++ [30] ignores e_i. **(b)** computes e_i based on low dimensional geometric relation between n_i and c. **(c)** also consider semantic relation between the center and the node point, but c has to be sampled on one of the points from the previous layer. **(d)**. Grid-GCN's geo-relation also includes the coverage weight. It pools a context feature f_{cxt} from all stored neighbors to provide a semantic reference in e_i computing.

allows us to calculate the semantic relation between the center and its node points.

- Even when group center is picked on a physical point, f_{cxt} is still a useful feature representation as it covers more points in the neighborhood, instead of only the points in the graph.

- Since we have already associated context points to its center voxel in CAGQ, there is no extra point query overhead. f_{cxt} is shared across all edge computation in a local graph, and the pooling is a light-weighted operation requiring no learnable weights, which introduces little computational overhead.

GCA module is summarized in Figure 4d, and the edge attention function can be model as

$$e = mlp(mlp_{geo}(\chi_c, \chi_i, w_i), mlp_{sem}(f_{cxt}, f_i)) \quad (6)$$

4. Analysis of CAGQ

To analyze the benefit of CAGQ, we test the occupied space coverage and the latency of different sampling/querying methods under different conditions on ModelNet40 [43]. Center sampling methods include Random Point Sampling (RPS), Farthest Point Sampling (FPS), our Random Voxel Sampling (RVS), and our Coverage-Aware Sampling (CAS). Neighbor querying methods include Ball Query, Cube query, and K-Nearest Neighbors. The conditions include different numbers of input points, node numbers in a point group, and numbers of point groups, which are denoted by N, K, and M. We summarize the qualitative and quantitative evaluation result in Table 2 and Figure 5. The reported occupied space coverage is calculated as the ratio between the number of voxels occupied by node points of all groups, and the number of voxels occupied by the original N points. Results under more conditions are presented in the supplementary.

4.1. Space Coverage

In Figure 5a, the centers sampled by RPS are concentrated in the areas with higher point density, leaving most space uncovered. In Figure 5b, FPS picks the points that are far away from each other, mostly on the edges of the 3D shape, which causes the gap between centers. In Figure 5c, our CAS optimizes the voxel selection and covers 75.2% of occupied space. Table 2 lists the percentage of space coverage by RPS, FPS, RVS, and CAS. CAS leads the space coverage in all cases (30 % more than RPS). FPS has no advantage over RVS when K is small.

The factors that benefit CAGQ in space coverage can be summarized as follows:

- Instead of sampling centers from N points, RVS samples center voxels from occupied space, therefore it is more resilient to point density imbalance (Figure 5).

- CAS further optimizes the result of RVS by conducting a greedy candidate replacement. Each replacement is guaranteed to result in better coverage.

- CAGQ stores the same number of points in each occupied voxel. The context points are more evenly distributed, so are the K node points picked from the context points. Consequently, the strategy reduces the coverage loss caused by density imbalance in a local area.

4.2. Time complexity

We summarize the time complexity of different methods in Table 1. The detailed deduction is presented in the supplementary. Table 2 shows the empirical results of latency. We see that our CAS is much faster than FPS and achieves $50\times$ speed-up. CAS + Cube Query can even outperform RPS + Ball Query when the size of the input point cloud is large. This is due to the higher neighborhood query speed. Because of better time complexity, RVS + k-NN leads the performance under all conditions and achieves $6\times$ speed-up over FPS + k-NN.

(a) Random Point Sampling   (b) Farthest Point Sampling   (c) Coverage-Aware Sampling
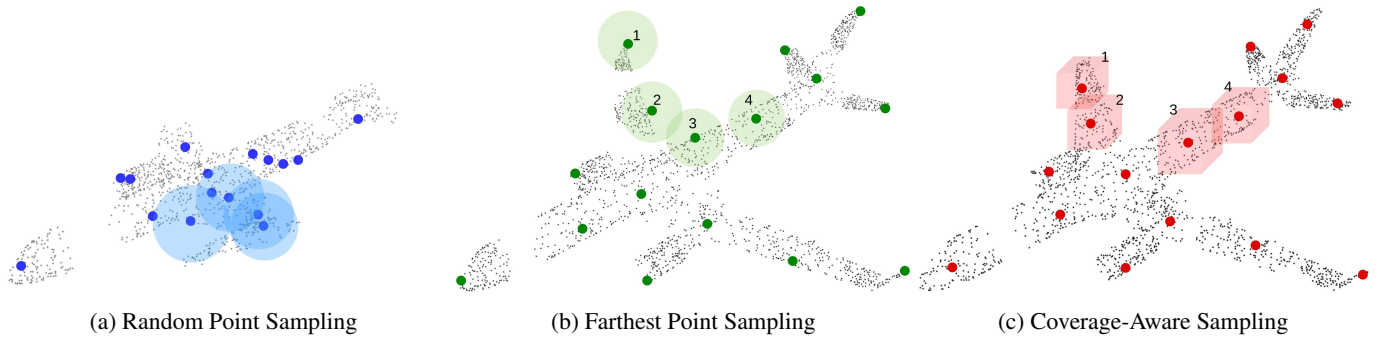
Figure 5: The visualization of the sampled group center and the queried node points by RPS, FPS, and CAS. The blue and green balls indicate Ball Query. The red squares indicate Cube Query. The ball and cube have the same volume. (a) RPS covers 45.6% of the occupied space, while FPS covers 65% and CAS covers 75.2%.

Sample	RPS	FPS[10]	RVS*	CAS*
centers	$O(N)$	$O(NlogN)$	$O(N)$	$O(N)$
Query	Ball Query	Cube Query*	k-NN[7]	CAGQ k-NN*
nodes	$O(MN)$	$O(MK)$	$O(MN)$	$O(Mn_v)$

Table 1: Time complexity: We sample M centers from N points and query K neighbors per center. We limit the maximum number of points in each voxel to n_v. In practice, $K < N$, and n_v is usually of the same magnitude to K. Approximate FPS algorithm can be $O(NlogN)$[9]. * indicates our methods. See the supplementary for deduction details.

5. Experiments

We evaluate Grid-GCN on multiple datasets: ModelNet10 and ModelNet40[43] for object classification, ScanNet[8] and S3DIS[1] for semantic segmentation. Following the convention of PVCNN [27], we report latency and performance in each level of accuracy. We collect the result of other models either from published papers or the authors. All the latency results are reported under the corresponding batch size and number of input points. All experiments are conducted on a single RTX 2080 GPU. Training details are listed in the supplementary.

5.1. 3D Object Classification

Datasets and settings We conduct the classification tasks on the ModelNet10 and ModelNet40 dataset[43]. ModelNet10 is composed of 10 object classes with 3991 training and 908 testing objects. ModelNet40 includes 40 different classes with 9843 training objects and 2468 testing objects. We prepare our data following the convention of PointNet[29], which uses 1024 points with 3 channels of spatial location as input. Several studies use normal [30, 16], octree [40], or kd-tree for input, and [26, 25] use voting for evaluation.

Evaluation To compare with different models with different levels of accuracy and speed, we train Grid-GCN with 4 different settings to balance performance and speed (Details are shown in section 5.3). The variants are in the

number of feature channels and the number of node points in a group in the first layer (see Table 6). The results are shown in Table 3. We report our results without voting. For all of the four settings, our Grid-GCN model not only achieves state-of-the-art performance on both ModelNet10 and ModelNet40 datasets, but has the best speed-accuracy trade-off. Although Grid-GCN uses the CAGQ module for data structuring, it has similar latency as PointNet which has no data structuring step while its accuracy is significantly higher than PointNet.

5.2. 3D Scene Segmentation

Dataset and Settings We evaluate our Grid-GCN on two large-scale point cloud segmentation datasets: ScanNet[8] and Stanford 3D Large-Scale Indoor Spaces (S3DIS) [1]. ScanNet consists of 1513 scanned indoor scene, and each voxel is annotated in 21 categories. We follow the experiment setting in [8] and use 1201 scenes for training, and 312 scenes for testing. Following the routine and evaluation protocol in PointNet++[30], we sample 8192 points during training and 3 spatial channels for each point. S3DIS contains 6 large-scale indoor areas with 271 rooms. Each point is labeled with one of 13 categories. Since area 5 is the only area that doesn't have overlaps with other areas, we follow [35, 23, 27] to train on area 1-4 and 6, and test on area 5. In each divided section, 4096 points are sampled for training, and we adopt the evaluation method from [23].

Evaluation We report the overall voxel labeling accuracy (OA) and the runtime latency for ScanNet[8]. We trained two versions of the Grid-GCN model, with a full model using $1 \times K$ node points and a compact model using $0.5 \times K$ node points. Results on are reported in Table 4.

Since the segmentation tasks generally use more input points than the classification model, our advantage of data structuring becomes outstanding. With the same amount of input points (32768) in a batch, Grid-GCN out-speed PointNet++ $4.5\times$ while maintaining the same level of accuracy. Compared with more sophisticated models such as PointCNN [23] and A-CNN [16], Grid-GCN is $25\times$

Center sampling	RPS	FPS	RVS*	CVS*	RPS	FPS	RVS*	CVS*	RPS	FPS	RVS*	CVS*
Neighbor querying	Ball	Ball	Cube	Cube	Ball	Ball	Cube	Cube	k-NN	k-NN	k-NN	k-NN
N K M	Occupied space coverage(%)				Latency (ms) with batch size = 1							
1024 8 8	12.3	12.9	13.1	**14.9**	**0.29**	0.50	0.51	0.74	0.84	0.85	**0.51**	0.77
1024 8 128	64.0	72.5	82.3	**85.6**	**0.32**	0.78	0.44	0.68	1.47	1.74	**0.52**	0.72
1024 128 32	60.0	70.1	61.0	**74.7**	**0.37**	0.53	0.96	1.18	22.23	21.08	**2.24**	2.74
1024 128 128	93.6	99.5	95.8	**99.7**	**0.38**	0.69	1.03	1.17	32.48	32.54	**6.85**	7.24
8192 8 64	19.2	22.9	22.1	**25.1**	**0.64**	1.16	0.66	0.82	1.58	1.80	**0.65**	0.76
8192 8 1024	82.9	**96.8**	92.4	94.4	0.81	4.90	**0.54**	0.87	1.53	5.36	**0.93**	0.97
8192 128 256	79.9	90.7	80.0	**93.5**	1.19	1.19	**1.17**	1.41	21.5	21.5	**15.19**	17.68
8192 128 1024	98.8	99.9	99.5	**100.0**	**1.22**	5.25	1.40	1.76	111.4	111.7	**24.18**	27.65
81920 32 1024	70.6	86.3	78.3	**91.6**	8.30	33.52	**3.34**	6.02	19.49	43.69	**8.76**	10.05
81920 32 10240	98.8	99.2	**100.0**	**100.0**	8.93	260.48	**4.22**	9.35	20.38	272.48	**9.65**	17.44
81920 128 1024	72.7	88.2	79.1	**92.6**	9.68	34.72	**4.32**	8.71	71.99	93.02	**50.7**	61.94
81920 128 10240	99.7	**100.0**	**100.0**	**100.0**	10.73	258.49	**5.83**	11.72	234.19	442.87	**69.02**	83.32

Table 2: Performance comparisons of data structuring methods, run on ModelNet40[43]. Center sampling methods include RPS, FPS, CAGQ's RVS and CAS. Neighbor querying methods include Ball Query, Cube query and K-Nearest Neighbors. Condition variables include N points, M groups and K neighbors per group. Occupied space coverage = num. of occupied voxels of queried points / num. of occupied voxels of the original N points.

Input (xyz as default)		ModelNet40		ModelNet10		latency
		OA	mAcc	OA	mAcc	(ms)
OA $\leqslant$ 91.5						
PointNet[29]	16×1024	89.2	86.2	-	-	**15.0**
SCNet[44]	16×1024	90.0	87.6	-		-
SpiderCNN[47]	8 × 1024	90.5	-	-	-	85.0
O-CNN[40]	octree	90.6	-	-	-	90.0
SO-net[22]	8 × 2048	90.8	87.3	**94.1**	**93.9**	-
Grid-GCN[1]	16×1024	**91.5**	**88.6**	93.4	92.1	15.9
OA $\leqslant$ 92.0						
3DmFVNet[3]	16×1024	91.6	-	95.2	-	39.0
PAT[48]	8 × 1024	91.7	-		-	88.6
Kd-net[15]	kd-tree	91.8	88.5	94.0	93.5	-
PointNet++[30]	16×1024	91.9	**90.7**		-	26.8
Grid-GCN[2]	16×1024	**92.0**	89.7	**95.8**	**95.3**	**21.8**
OA > 92.0						
DGCNN[41]	16×1024	92.2	90.2		-	89.7
PCNN[2]	16×1024	92.3	-	94.9	-	226.0
Point2Seq[24]	16×1024	92.6	-			-
A-CNN[16]	16×1024	92.6	90.3	95.5	95.3	68.0
KPConv[36]	16×6500	92.7	-	-	-	125.0
Grid-GCN[3]	16×1024	92.7	90.6	96.5	95.7	**26.2**
Grid-GCN^{full}	16×1024	**93.1**	**91.3**	**97.5**	**97.4**	42.2

Table 3: Results on ModelNet10 and ModelNet40[43]. Our full model achieves the state-of-the-art accuracy. With model reduction, our compact models Grid-GCN[1−3] also out speed other models. We discuss their details in the ablation studies.

and 12× faster, respectively, while achieving the state-of-the-art accuracy. Remarkably, Grid-GCN can run as fast as **50** to **133** FPS with state-of-the-art performance, which is desired in real-time applications. A popular model

Input (xyz as default)		OA	latency (ms)
OA < 84.0			
PointNet[29]	8 × 4096	73.9	20.3
OctNet[31]	volume	76.6	-
PointNet++[30]	8 × 4096	83.7	72.3
Grid-GCN$_{(0.5×K)}$	4 × 8192	**83.9**	**16.6**
OA $\geqslant$ 84.0			
SpecGCN[37]	-	84.8	-
PointCNN[23]	12×2048	85.1	250.0
Shellnet[50]	-	85.2	-
Grid-GCN$_{(1×K)}$	4 × 8192	**85.4**	**20.8**
A-CNN[16]	1 × 8192	**85.4**	92.0
Grid-GCN$_{(1×K)}$	1 × 8192	**85.4**	**7.48**

Table 4: Results on ScanNet[8]. Grid-GCN achieves 10× speed-up on average over other models. Under batch size of 4 and 1, we test our model with $1×K$ neighbor nodes. A compact model with $0.5 × K$ is also reported.

MinkowskiNet[5] doesn't report the overall accuracy, therefore we don't put it in the table. But its github example shows a latency of 103ms on Scannet.

We show the quantitative results on S3DIS in Table 5 and visual result in Figure 6. Our compact version of Grid-GCN is generally 4× to 14× faster than other models with data structuring. Notably, even compared with PointNet that has no data structuring at all, we are still 1.6× faster while achieves 12% performance gain in mIOU. For our full model, we are still the fastest and achieve 2× speed-up over PVCNN++[27], a state-of-the-art study focusing on speed improvement.
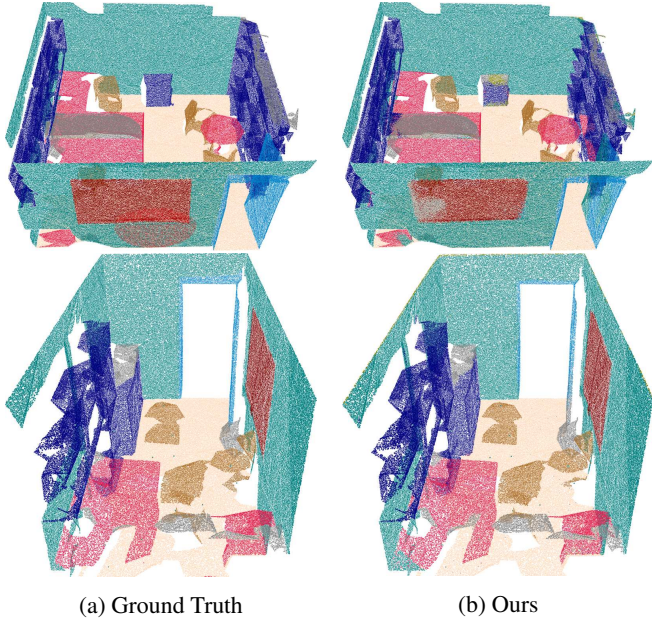
(a) Ground Truth   (b) Ours

Figure 6: Semantic segmentation results on S3DIS [1] area 5.

Input (xyzrgb as default)		mIOU	OA	latency(ms)
mIOU< 54.0				
PointNet[29]	8×4096	41.09	-	20.9
DGCNN[41]	8×4096	47.94	83.64	178.1
SegCloud[35]	-	48.92	-	-
RSNet[14]	8×4096	51.93	-	111.5
PointNet++[30]	8×4096	52.28		
DeepGCNs[21]	1×4096	52.49	-	45.63
TanConv[34]	8×4096	52.8	85.5	-
Grid-GCN$_{(0.5 \times Ch)}$	8×4096	**53.21**	**85.61**	**12.9**
mIOU> 54.0				
3D-UNet[6]	8×96^3 volume	54.93	86.12	574.7
PointCNN[23]	-	57.26	85.91	-
PVCNN++[27]	8×4096	57.63	86.87	41.1
Grid-GCN$_{(1 \times Ch)}$	8×4096	**57.75**	**86.94**	**25.9**

Table 5: Results on S3DIS[1] area 5. Grid-GCN is on average $8\times$ faster than other models. We halve the output channels of GridConv for Grid-GCN$_{(0.5 \times Ch)}$.

5.3. Ablation Studies

In the experiment on ModelNet10 and ModelNet40[43], our full model has 3 GridConv layers. As shown in Table 6, we conduct reductions on the number of the output feature channels from GridConv layers, the number of nodes K in the first GridConv layer, and whether to use Grid context pooling and coverage weight. On one hand, reducing the number of channels from Grid-GCN^{full} gives Grid-GCN^3 37% speed-up. On the other hand, reducing K and removing Grid context pooling from Grid-GCN^3 doesn't give Grid-GCN^2 much speed benefit but incurs a loss on accu-

	K	Channels	Pooling	Weight	OA	latency
Grid-GCN^0	32	(32,64,256)	No	No	91.1	15.4ms
Grid-GCN^1	32	(32,64,256)	No	Yes	91.5	15.9ms
Grid-GCN^2	32	(64,128,256)	No	Yes	92.0	21.8ms
Grid-GCN^3	64	(64,128,256)	Yes	Yes	92.7	26.2ms
Grid-GCN^{full}	64	(128,256,512)	Yes	Yes	93.1	42.2ms

Table 6: Ablation studies on ModelNet40[43]. Our models have 3 layers of GridConv. K is the number of node points in the first GridConv. We also change the number of the output feature channels from these 3 layers. Grid context pooling (shorted as pooling here) are also removed for Grid-GCN$^{0-2}$. Grid-GCN^0 also removes coverage weight in edge relation.

racy. This demonstrates the efficiency and effectiveness of CAGQ and Grid context pooling. Coverage weight is useful as well because it introduces little overhead in latency but increases the overall accuracy.

5.3.1 Scalability Analysis

Num. of points (N)	2048	4096	16384	40960	81920
Num. of clusters (M)	512	1024	2048	4096	8192
PointNet++	4.7	8.6	19.9	64.6	218.9
Grid-GCN	4.3	4.7	8.1	12.3	19.8

Table 7: Inference time (ms) on ScanNet[8] under different scales. We compare Grid-GCN with PoinNet++[30] on different numbers of input points per scene. The batch size is 1. M is the number of point groups on the first network layer.

We also test our model's scalability by gradually increasing the number of input points on ScanNet [8]. We compare our model with PointNet++ [30], one of the most efficient point-based method. We report the results in Table 7. Under the setting of 2048 points, the latency of two models are similar. However, when increasing the input point from 4096 to 81920, Grid-GCN achieves up to $11\times$ speed-up over PointNet++, which shows the dominating capability of our model in processing large-scale point clouds.

6. Conclusion

In this paper, we propose Grid-GCN for fast and scalable point cloud learning. Grid-GCN achieves efficient data structuring and computation by introducing Coverage-Aware Grid Query (CAGQ). CAGQ drastically reduces data structuring cost through voxelization and provides point groups with complete coverage of the whole point cloud. A graph convolution module Grid Context Aggregation (GCA) is also proposed to incorporate the context features and coverage information in the computation. With both modules, Grid-GCN achieves state-of-the-art accuracy and speed on various benchmarks. Grid-GCN, with its superior performance and unparalleled efficiency, can be used in large-scale real-time point cloud processing applications.

References

[1] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, Feb. 2017. 2, 6, 8

[2] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*, 2018. 2, 7

[3] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*, 3(4):3145–3152, 2018. 7

[4] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016. 2

[5] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 7

[6] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016. 2, 3, 4, 8

[7] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967. 6

[8] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 2, 6, 7, 8

[9] Y Eldar. *Irregular image sampling using the voronoi diagram*. PhD thesis, M. Sc. thesis, Technion-IIT, Israel, 1992. 2, 6

[10] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997. 2, 6

[11] Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3254–3263, 2019. 2

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[13] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 2

[14] Qiangui Huang, Weiyue Wang, and Ulrich Neumann. Recurrent slice networks for 3d segmentation of point clouds.

[15] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 863–872, 2017. 2, 7

[16] Artem Komarichev, Zichun Zhong, and Jing Hua. A-cnn: Annularly convolutional neural networks on point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7421–7430, 2019. 6, 7

[17] Shiyi Lan, Ruichi Yu, Gang Yu, and Larry S Davis. Modeling local geometric structure of 3d point clouds using geo-cnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 998–1008, 2019. 2, 3, 4

[18] Loic Landrieu and Mohamed Boussaha. Point cloud oversegmentation with graph-structured deep metric learning. *arXiv preprint arXiv:1904.02113*, 2019. 2

[19] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018. 2

[20] Truc Le and Ye Duan. Pointgrid: A deep network for 3d shape understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9204–9214, 2018. 2

[21] Guohao Li, Matthias Müller, Guocheng Qian, Itzel C Delgadillo, Abdulellah Abualshour, Ali Thabet, and Bernard Ghanem. Deepgcns: Making gcns go as deep as cnns. *arXiv preprint arXiv:1910.06849*, 2019. 2, 8

[22] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018. 2, 7

[23] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pages 820–830, 2018. 2, 6, 7, 8

[24] Xinhai Liu, Zhizhong Han, Yu-Shen Liu, and Matthias Zwicker. Point2sequence: Learning the shape representation of 3d point clouds with an attention-based sequence to sequence network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8778–8785, 2019. 7

[25] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. Densepoint: Learning densely contextual representation for efficient point cloud processing. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5239–5248, 2019. 6

[26] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019. 2, 3, 4, 6

[27] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. *arXiv preprint arXiv:1907.03739*, 2019. 2, 6, 7, 8

[28] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition.

In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015. 1, 2

[29] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas 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*, pages 652–660, 2017. 1, 2, 6, 7, 8

[30] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017. 1, 2, 4, 5, 6, 7, 8

[31] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017. 2, 7

[32] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017. 2

[33] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2530–2539, 2018. 2

[34] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3887–3896, 2018. 8

[35] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *2017 International Conference on 3D Vision (3DV)*, pages 537–547. IEEE, 2017. 6, 8

[36] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. *arXiv preprint arXiv:1904.08889*, 2019. 2, 7

[37] Chu Wang, Babak Samari, and Kaleem Siddiqi. Local spectral graph convolution for point set feature learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–66, 2018. 1, 2, 3, 7

[38] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems*, volume 1, pages 10–15607, 2015. 2

[39] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10296–10305, 2019. 2, 3, 4

[40] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM Transactions on Graphics (SIGGRAPH)*, 36(4), 2017. 6, 7

[41] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):146, 2019. 1, 2, 3, 4, 7, 8

[42] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019. 2

[43] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 2, 5, 6, 7, 8

[44] Saining Xie, Sainan Liu, Zeyu Chen, and Zhuowen Tu. Attentional shapecontextnet for point cloud recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4606–4615, 2018. 7

[45] Christopher B Choy Danfei Xu, JunYoung Gwak, and Kevin Chen Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. *arXiv preprint arXiv:1604.00449*, 2016. 1

[46] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 492–502. Curran Associates, Inc., 2019. 1

[47] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102, 2018. 1, 2, 3, 4, 7

[48] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3323–3332, 2019. 2, 7

[49] Kuangen Zhang, Ming Hao, Jing Wang, Clarence W de Silva, and Chenglong Fu. Linked dynamic graph cnn: Learning on point cloud via linking hierarchical features. *arXiv preprint arXiv:1904.10014*, 2019. 2, 3, 4

[50] Zhiyuan Zhang, Binh-Son Hua, and Sai-Kit Yeung. Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. *arXiv preprint arXiv:1908.06295*, 2019. 7

[51] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018. 1, 2