Grid-GCN for Fast and Scalable Point Cloud Learning

Qiangeng Xu\textsuperscript{1} Xudong Sun\textsuperscript{2} Cho-Ying Wu\textsuperscript{1} Panqu Wang\textsuperscript{2} Ulrich Neumann\textsuperscript{1}
\textsuperscript{1}University of Southern California \{qiangengx,choyingw,uneumann\}@usc.edu \textsuperscript{2}Tusimple, Inc \{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 interrelation. 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 $50$fps on ScanNet using $81920$ points as input. The supplementary\textsuperscript{1} and the code\textsuperscript{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 \cite{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 structures 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\cite{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, PointNet\cite{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\cite{30, 47, 41, 37, 46} apply a downsampling strategy at each layer to aggregate information.

\begin{figure}[h!]
\centering
\includegraphics[width=\textwidth]{grid-gcn.png}
\caption{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.}
\end{figure}

\begin{flushleft}
\textsuperscript{1}https://xharlie.github.io/papers/GGCN_supCamReady.pdf
\textsuperscript{2}https://github.com/xharlie/Grid-GCN
\end{flushleft}
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 $O(N \log N)$. 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
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 GridConv 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 \(Vid(u, v, w) = floor\left(\frac{u}{v_x}, \frac{v}{v_y}, \frac{w}{v_z}\right)\). Here we only store up to \(n_c\) points in each voxel.

Let \(O_c\) 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 $\lambda$ 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 $O_c$ 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:

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

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

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

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

$$\tilde{f}_{c,i} = e(\chi_i, f_i) * M(f_i) \quad (4)$$

$$\tilde{f}_c = A(\{\tilde{f}_{c,i}\}, i \in 1, ..., K) \quad (5)$$

where $\tilde{f}_{c,i}$ is the contribution from a node, and $\chi_i$ is the xyz location of the node. $M$ is a multi-layer perceptron (MLP), $e$ is the edge attention function, and $A$ 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 following improvements to better fit into our network architecture (Figure 4):

**Coverage Weight** Previous studies [47, 26, 17, 41, 49] use $\chi_c$ of the center and $\chi_i$ 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
allows us to calculate the semantic relation between the center and its node points.

- Even when group center is picked on a physical point, \( f_{ext} \) 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. \( f_{ext} \) 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

\[
e = mlp(mlp_{geo}(\chi_c, \chi_i, w_i), mlp_{sem}(f_{ext}, f_i))
\]  

(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× 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× speed-up over FPS + k-NN.
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 scenes, 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× while maintaining the same level of accuracy. Compared with more sophisticated models such as PointCNN [23] and A-CNN [16], Grid-GCN is 25×
### Table 2: Performance comparisons of data structuring methods, run on ModelNet40.[43]

Table 2 shows the performance comparisons of data structuring methods, run on ModelNet40. 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.

<table>
<thead>
<tr>
<th>Center sampling</th>
<th>RPS</th>
<th>FPS</th>
<th>CVS*</th>
<th>Occupied space coverage(%)</th>
<th>Latency (ms) with batch size = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ball</td>
<td>Ball</td>
<td>Cube</td>
<td>Cube</td>
<td>RPS</td>
</tr>
<tr>
<td>N</td>
<td>K</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>8</td>
<td>8</td>
<td>12.3</td>
<td>12.9</td>
<td>13.1</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>128</td>
<td>64.0</td>
<td>72.5</td>
<td>82.3</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>32</td>
<td>60.0</td>
<td>70.1</td>
<td>61.0</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>128</td>
<td>93.6</td>
<td>99.5</td>
<td>95.8</td>
</tr>
<tr>
<td>8192</td>
<td>8</td>
<td>64</td>
<td>19.2</td>
<td>22.9</td>
<td>22.1</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1024</td>
<td>82.9</td>
<td>96.8</td>
<td>92.4</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>256</td>
<td>79.9</td>
<td>90.7</td>
<td>80.0</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>1024</td>
<td>98.8</td>
<td>99.9</td>
<td>99.5</td>
</tr>
<tr>
<td>81920</td>
<td>32</td>
<td>1024</td>
<td>70.6</td>
<td>86.3</td>
<td>78.3</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>10240</td>
<td>98.8</td>
<td>99.2</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>1024</td>
<td>72.7</td>
<td>88.2</td>
<td>79.1</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>10240</td>
<td>99.7</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Input (xyz as default)</th>
<th>OA</th>
<th>mAcc</th>
<th>ModelNet40</th>
<th>ModelNet10</th>
<th>latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OA</td>
<td>mAcc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OA</td>
<td></td>
<td>OA</td>
<td>mAcc</td>
<td>latency (ms)</td>
</tr>
<tr>
<td>PointNet[29]</td>
<td>16×1024</td>
<td>89.2</td>
<td>86.2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SCNet[44]</td>
<td>16×1024</td>
<td>90.0</td>
<td>87.6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SpiderCNN[47]</td>
<td>8×1024</td>
<td>90.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>O-CNN[40]</td>
<td>octree</td>
<td>90.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SO-net[22]</td>
<td>8×2048</td>
<td>90.8</td>
<td>87.3</td>
<td>94.1</td>
<td>93.9</td>
</tr>
<tr>
<td>Grid-GCN(^1)</td>
<td>16×1024</td>
<td>91.5</td>
<td>88.6</td>
<td>93.4</td>
<td>92.1</td>
</tr>
<tr>
<td>D3mFVNet[3]</td>
<td>16×1024</td>
<td>91.6</td>
<td>-</td>
<td>95.2</td>
<td>-</td>
</tr>
<tr>
<td>PAT[48]</td>
<td>8×1024</td>
<td>91.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Kd-net[15]</td>
<td>kd-tree</td>
<td>91.8</td>
<td>88.5</td>
<td>94.0</td>
<td>93.5</td>
</tr>
<tr>
<td>PointNet++[30]</td>
<td>16×1024</td>
<td>91.9</td>
<td>90.7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Grid-GCN(^2)</td>
<td>16×1024</td>
<td>92.0</td>
<td>89.7</td>
<td>95.8</td>
<td>95.3</td>
</tr>
<tr>
<td>DGCNN[41]</td>
<td>16×1024</td>
<td>92.2</td>
<td>90.2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PCNN[2]</td>
<td>16×1024</td>
<td>92.3</td>
<td>-</td>
<td>94.9</td>
<td>-</td>
</tr>
<tr>
<td>Point2Seq[24]</td>
<td>16×1024</td>
<td>92.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A-CNN[16]</td>
<td>16×1024</td>
<td>92.6</td>
<td>90.3</td>
<td>95.5</td>
<td>95.3</td>
</tr>
<tr>
<td>KPConv[36]</td>
<td>16×6500</td>
<td>92.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Grid-GCN(^3)</td>
<td>16×1024</td>
<td>92.7</td>
<td>90.6</td>
<td>96.5</td>
<td>95.7</td>
</tr>
<tr>
<td>Grid-GCN(^*)</td>
<td>16×1024</td>
<td>93.1</td>
<td>91.3</td>
<td>97.5</td>
<td>97.4</td>
</tr>
</tbody>
</table>

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.
Table 5: Results on S3DIS[1] area 5.

<table>
<thead>
<tr>
<th>Input (xyzrgb as default)</th>
<th>mIOU</th>
<th>OA</th>
<th>latency(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet[29]</td>
<td>41.09</td>
<td>-</td>
<td>20.9</td>
</tr>
<tr>
<td>DGCNN[41]</td>
<td>47.94</td>
<td>83.64</td>
<td>178.1</td>
</tr>
<tr>
<td>SegCloud[35]</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RSNet[14]</td>
<td>51.93</td>
<td>-</td>
<td>111.5</td>
</tr>
<tr>
<td>PointNet++[30]</td>
<td>52.28</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DeepGCNs[21]</td>
<td>52.49</td>
<td>-</td>
<td>45.63</td>
</tr>
<tr>
<td>TanConv[34]</td>
<td>52.8</td>
<td>85.5</td>
<td>-</td>
</tr>
<tr>
<td>Grid-GCN((0.5 \times Ch)_k)</td>
<td>53.21</td>
<td>85.61</td>
<td>12.9</td>
</tr>
</tbody>
</table>

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

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\(f_{full}\) 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\(^3\) much speed benefit but incurs a loss on accuracy. This demonstrates the efficiency and effectiveness of CAQQ 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

<table>
<thead>
<tr>
<th>Num. of points (N)</th>
<th>2048</th>
<th>4096</th>
<th>16384</th>
<th>40960</th>
<th>81920</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. of clusters (M)</td>
<td>512</td>
<td>1024</td>
<td>2048</td>
<td>4096</td>
<td>8192</td>
</tr>
<tr>
<td>PointNet++</td>
<td>4.7</td>
<td>8.6</td>
<td>19.9</td>
<td>64.6</td>
<td>218.9</td>
</tr>
<tr>
<td>Grid-GCN</td>
<td>4.3</td>
<td>4.7</td>
<td>8.1</td>
<td>12.3</td>
<td>19.8</td>
</tr>
</tbody>
</table>

Table 7: Inference time (ms) on ScanNet[8] under different scales. We compare Grid-GCN with PointNet++[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× 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 (CAQQ). CAQQ 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


\[
\text{In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 922--928. IEEE, 2015. 1, 2.}
\]


[39] Lei Wang, Yuchun Huang, Yaolin Hou, Shennan 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


