Towards Efficient Graph Convolutional Networks for Point Cloud Handling

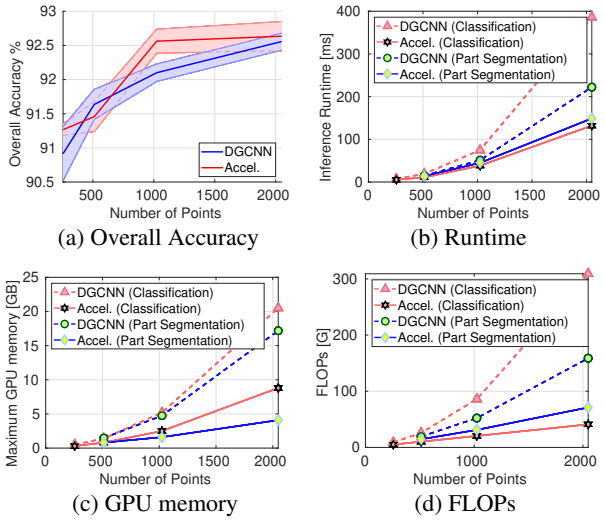
Yawei Li[1*], He Chen[2*], Zhaopeng Cui[3], Radu Timofte[1], Marc Pollefeys[1,4],
Gregory Chirikjian[2,5], Luc Van Gool[1,6]
[1]ETH Zürich, [2]Johns Hopkins University, [3]State Key Lab of CAD&CG, Zhejiang University,
[4]Microsoft, [5]National University of Singapore, [6]KU Leuven, Belgium
yawei.li@vision.ee.ethz.ch, hchen136@jhu.edu, zhpcui@gmail.com

Abstract

We aim at improving the computational efficiency of graph convolutional networks (GCNs) for learning on point clouds. The basic graph convolution that is composed of a K-nearest neighbor (KNN) search and a multilayer perceptron (MLP) is examined. By mathematically analyzing the operations there, two findings to improve the efficiency of GCNs are obtained. (1) The local geometric structure information of 3D representations propagates smoothly across the GCN that relies on KNN search to gather neighborhood features. This motivates the simplification of multiple KNN searches in GCNs. (2) Shuffling the order of graph feature gathering and an MLP leads to equivalent or similar composite operations. Based on those findings, we optimize the computational procedure in GCNs. A series of experiments show that the optimized networks have reduced computational complexity, decreased memory consumption, and accelerated inference speed while maintaining comparable accuracy for learning on point clouds.

1. Introduction

Recently, graph convolutional networks (GCN) [8, 2, 10, 53, 43, 58, 48] have achieved state-of-the-art performances in 3D representation learning on point clouds for classification [35, 36], part segmentation [3], semantic segmentation [49, 19], and surface reconstruction [14]. A typical GCN is composed of a stack of multilayer perceptrons (MLPs) that progressively learn a hierarchy of deep features. For a better modelling of the locality on point clouds, neighborhood information gathering modules are placed before MLPs. A certain point gathers information from its neighbors and propagates its information to them. The neighbors can be predefined (*i.e.*, borrowed from an initial mesh in Point2Mesh [14]) or more commonly established by K-nearest neighbor (KNN) search on point clouds (static GCN [36, 23]) or on the feature representation (dynamic GCN [49, 54]).



(a) Overall Accuracy (b) Runtime (c) GPU memory (d) FLOPs

Figure 1: Comparison between a representative GCN and the accelerated version in this paper. (a) The overall accuracy for point cloud classification. Mean and Variance reported for 5 runs. The (b) runtime, (c) GPU memory consumption, and (d) FLOPs of the original GCN explodes with an increasing number of points. By contrast, the optimized network can achieve a significant reduction of computational resources without a drop in accuracy.

Yet, this design faces several technical challenges. **Firstly**, the computational cost grows quadratically with the number of points [38, 37]. The problem is exacerbated when KNN search is conducted in a high-dimensional feature space. **Secondly**, the graph feature gathering operation expands the dimension of the resultant features. Consider a point cloud with N points and d coordinates. The dimension of the tensor grows from $N \times d$ to $N \times K \times d$ after the K graph feature gathering operation, where K is the number of neighbors. Then the same operation is applied to the expanded tensor with repeated entries, which leads to

*Co-first author.

redundant computations. **Thirdly**, due to the computational complexity and the expanded features, the GPU memory required for GCN computations explodes when the number of processed points increases. The inference speed also slows down drastically.

As shown in Fig. 1, each time the number of processed points doubles, the computational complexity, inference time, and consumed GPU memory of the examined GCN almost quadruple. Thus, the aim of this paper is to analyze the basic operations in GCNs and seek opportunities to build efficient GCNs for learning on point clouds. Compared with the representative GCN in Fig. 1, the computationally optimized GCN in this paper reduces the computational burden and accelerates the inference. This significant improvement relies on the following two findings.

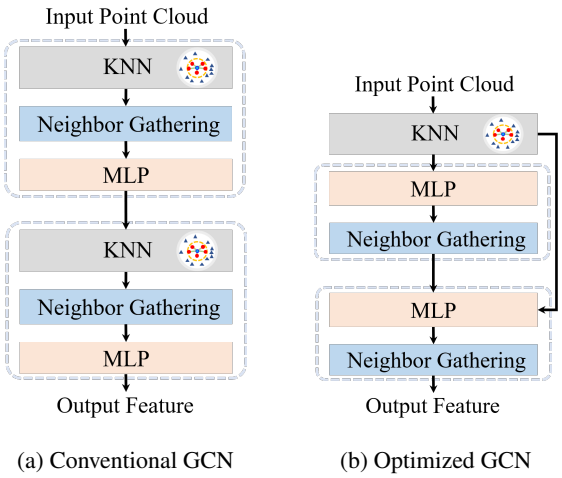
Finding 1. *The local geometric structure information of 3D representations propagates smoothly across the aforementioned multilayer GCN that relies on KNN search for graph feature gathering.*

This finding is supported by the mathematical analysis of the distances between two points before and after one layer of an MLP. In Sec. 4.2, we show that the distance between two points after one layer of MLP is upper bounded by the neighborhood distance and lower bounded by the neighborhood centroid distance between the corresponding points before the MLP. This means that across a GCN the distance between two points in the feature space does not abruptly change. Thus, it is not necessary to conduct KNN search every time a neighbor retrieval is needed in MLPs. Instead, a couple of MLPs (referred to as shareholder MLP) can share the results of the same KNN search. Moreover, to ensure a progressively enlarged receptive field across the shareholder MLPs, a larger pool of neighbors can be kept from the first KNN search. Each time neighbor retrieval is needed, the neighbors are sampled from the pool. The shareholder MLPs in the shallower layers can only sample from the near neighbors while the deep shareholders have the chance to sample from far-away neighbors.

Finding 2. *Shuffling the order of the graph feature gathering operation and the MLP used for feature extraction leads to equivalent or similar composite operations for GCNs.*

This finding is also supported by a general analysis in Sec. 4.3. As said, in existing GCNs, the graph feature gathering operation happens before the MLP and expand the dimension of the features. By moving the feature extracting MLP before the graph feature gathering operation, the MLP is conducted merely on the non-expanded feature tensors. And this leads to a significant reduction in computations.

The two findings directly lead to the proposed change in computational procedure as shown in Fig. 2, which reduces the computational complexity and accelerates the inference of the GCNs. Here, the proposed techniques are



(a) Conventional GCN (b) Optimized GCN

Figure 2: Comparison between (a) conventional GCN and (b) the optimized GCN in this paper. Instead of calling KNN search for each graph convolution, we enforce several graph convolutions to share the same KNN search with progressively enlarged receptive fields. The shuffling of graph feature gathering and MLP avoids the expansion of features, which leads to accelerated computation in the MLP.

applied to four representative GCNs [49, 23, 14, 54]. It is shown that they can improve the efficiency of existing GCNs significantly, indeed. For example, for ModelNet40 point cloud classification with 2048 points, compared with the original DGCNN, the accelerated version is about $\times\mathbf{3}$ **times faster**, reduces GPU memory by **57.1%** and computation by **86.7%** without loss of accuracy. More results are shown in Sec. 5. Thus, the contributions of this paper can be summarized as follows.

1. Starting with the analysis of basic operations in representative GCNs, two theorems enabling their acceleration are proved.

2. Based on the proved theorems, two strategies for shuffling operations are proposed to specifically improve the time and memory efficiency of existing GCNs.

3. Extensive experiments on four GCNs for four point cloud learning tasks are carried out, to validate the efficiency of the proposed method. It is demonstrated that both the inference time and memory consumption decreased significantly.

2. Related Work

The last years have seen a trend of applying deep neural networks to 3D representations. In this process, computation-efficient network design plays an important role. We briefly summarize closely related contributions.

Deep Learning for 3D Point Clouds. With the easier access to large scale 3D scanned data, convolutional neural networks have been extended from learning features in 2D images [15, 40, 11, 55, 4] to learning from graph data [52, 9, 22, 7, 5] and 3D point clouds [35, 36, 59, 12]. Existing methods can be roughly categorized into voxel-based methods, point-based methods, and voxel-point-mixture methods [42]. Voxel-based methods [33, 34] leverage the architecture of 3D CNNs and apply it to rasterised 3D space. While point-based methods [39, 60, 21, 60] target at an explicit representation and directly operate on graphs. PointNet [35] pioneered the point-based methods by designing a network architecture based on MLP that directly consumes point clouds while respecting the permutation invariance of input data. However, the design of PointNet neglects local structures. Targeting at improving this drawback, PointNet++ [36] introduces a hierachical architecture that recursively calls PointNet on a nested partitioning of input point set. Another way to achieve improvements is via a Dynamic Graph CNN [49], which takes topological information into consideration by defining edge convolution operations.

Efficient Network Design plays an increasingly important role in computer vision. Seminal contributions include GoogLeNet [44], SqueezeNet [20], MobileNets [18, 41], and ShuffleNets [56, 32], which reduce model complexity by designing computational efficient modules. Other techniques in efficient design include network pruning [16, 31, 30, 25, 27, 28], low-rank filter approximation [57, 26], network quantization [13, 61, 24], and knowledge distillation [17, 46]. Computational efficiency enables running well performing neural networks on mobile devices, as well as processing more and more complicated 3D/4D scenes on powerful computers. Targeting at processing 3D/4D scenes with higher performance on the same computational resource, Vote3D [47] and FPNN [29] propose to improve efficiency by dealing with the sparsity problem. Minkowski Engine [6] proposes sparse convolution which uses a hash table for indexing during the convolution process. These methods are designed for improving the efficiency of voxel-based methods. Other efficient designs for point-based neural networks delve into the basic operations including convolution, pooling, and unpooling [19, 22, 9, 52, 51], in the same vein as our work.

3. Notations and Preliminaries

To formally formulate the problem, a couple of concepts are defined in this section. In the following Definition 1 and Definition 2, the neighbors of two points $\mathbf{x}_i$ and $\mathbf{x}_j$ are sorted according to the distance relative to these points, respectively.

Definition 1 (Neighborhood Distance) *Consider two points in a point set $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{S}$. Each of them is equipped with a neighborhood of points derived from KNN search,* i.e.*, $\mathcal{N}_i$, $\mathcal{N}_j$. The neighborhood distance between the two points is the sum of distances between their neighbors,*

$$\mathcal{D}_{\mathcal{N}}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{K} \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2. \tag{1}$$

Definition 2 (Neighborhood Centroid Distance) *The neighborhood centroid distance of two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined by the distance between the centroids of their K-nearest neighbors,* i.e.*,*

$$\mathcal{D}_{\mathcal{NC}} = \|\frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_i^k - \frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_j^k\|_2^2, \tag{2}$$

where $\frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_i^k$ denotes the centroid of the neighbors.

The Neighborhood Distance indicates the distance between two points. That is, two points with smaller Neighborhood Distance are highly likely to be closer to each other compared to those with larger Neighborhood Distance. Similarly, the Neighborhood Centroid Distance is also a metric that reflects the closeness of two points.

Definition 3 (Graph and Subgraph) *A graph is defined by a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges that defines the connectivity between vertices. A subgraph of a graph $\mathcal{G}$ is defined by the pair $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$, where $\mathcal{V}_i \subseteq \mathcal{V}$, and $\mathcal{E}_i \subseteq \mathcal{E}$.* A graph $\mathcal{G}$ can be defined on point clouds and meshes. A subgraph $\mathcal{G}_i$ captures the local connectivity on the 3D representation and is constructed slightly differently for point clouds and meshes. For a point cloud, the vertices of the subgraph include a point and its K-nearest neighbors and the edge connects the center point and the neighbors. For a mesh, the edge set $\mathcal{E}_i$ contains an edge and its 4 1-ring neighbors [14], and the vertex set $\mathcal{V}_i$ contains the 4 associated vertices.

Definition 4 (Graph Convolution) *Graph convolution is a family of operations that extract higher-level features from the lower-level ones by propagating information between vertices $\mathcal{V}$ or edges $\mathcal{E}$ of the defined graph $\mathcal{G}$. Graph convolution can be defined in terms of the subgraphs,* i.e.*,*

$$\mathbf{x'}_i = g(\mathcal{G}_i; \mathbf{\Theta}) = \sum_{\mathbf{e}_i^k \in \mathcal{E}_i} h(\mathbf{x}_i^k; \mathbf{\Theta}_k), \tag{3}$$

where $\mathbf{x}_i^k \in \mathbb{R}^d$ is a vector that encodes the feature on the edge $\mathbf{e}_i^k \in \mathcal{E}_i$, $\mathbf{x'}_i \in \mathbb{R}^M$ is the output feature, $\mathbf{\Theta} = \{\mathbf{\Theta}_k \in \mathbb{R}^{d\times M} | k = 1, 2, \cdots, K\}$ is the ensemble of the trainable parameters and is the same for all of the subgraphs $\mathcal{G}_i$.

The function $h(\cdot)$ transforms a d-dimensional input feature into a M-dimensional vector. It denotes an MLP, which in turn can be implemented as convolution operation. In this specific case, the aggregation function is a summation denoted by $\sum$. Generally, the aggregation function is a symmetric function (*e.g.*, $\sum$ or $\max$) that does not depend on

the order of the edges. A stack of graph convolutions and other operations such as pooling constitute a GCN.

Notation. In this paper, N represents the number of points, d represents the dimensionality of the latent space features, K represents the number of neighbors for each point, and M represents the dimensionality of the intermediate output features.

4. Methodology

In this section, the basic operations in GCNs, *i.e.*, KNN search and MLP in graph convolution, are analyzed. Two theorems about the properties of the two operations are proposed. Building on those, a simplified computational procedure for KNN search and MLP is introduced, which improves the computational efficiency of existing GCNs.

4.1. Computational complexity analysis in GCN

In state-of-the-art GCNs, KNN search is usually conducted to define the neighborhood, followed by an MLP. The computational complexity of the two operations are analyzed and the simplification methods are presented.

Proposition 1 *The ratio of the computational complexities of KNN search and MLP in a graph convolution is* $\gamma = \frac{N}{KM}$.

Assume that the point cloud is represented by a $N \times d$ matrix $\mathbf{X}$. To compute the K-nearest neighbors of all the points, a pairwise comparison between the points is conducted, *i.e.*, $\mathbf{D} = \mathbf{X}\mathbf{X}^T$. Then for each point, the indices of the K-nearest neighbors are kept and used to extract the graph feature, which results in a 3D tensor with a dimension of $N \times K \times d$. Then an MLP implemented as convolution with kernel size 1×1 and output channel M is conducted on the graph feature. The pairwise comparison and the 1×1 convolution are the computation-intensive parts.

The computational complexity, namely the number of multiplications in the pairwise comparisons is $\mathcal{C}_{nn} = dN^2$. And the computational complexity of the 1×1 convolution is $\mathcal{C}_{conv} = dMKN$. Thus, the ratio between the two complexity terms is

$$\gamma = \frac{\mathcal{C}_{nn}}{\mathcal{C}_{conv}} = \frac{dN^2}{dMKN} = \frac{N}{KM} \tag{4}$$

Compared with the number of nearest neighbors K and the output channel dimensionality M, the number of points in a point cloud could vary drastically. When N is small, the pairwise computation load is relatively small and even negligible. But when the point cloud grows huge, the computational load of this pairwise comparison could become dominant. This analysis shows the necessity of simplifying KNN search in GCNs.

4.2. Propagation of point adjacency

In the following, we investigate how local geometric structure information propagates within the GCN, by analyzing the adjacency property of points before and after graph convolution. This new perspective motivates us to rethink the necessity of frequent KNN callings in GCNs, as already hinted at earlier. It results in a simplification and acceleration of the adjacency assessment in GCNs. We consider a special case of the graph convolution in Eqn. 3 in the following form

$$\mathbf{x}'_i = [\mathbf{x}'_{i1}, \cdots, \mathbf{x}'_{im}, \cdots, \mathbf{x}'_{iM}], \tag{5}$$

$$\mathbf{x}'_{im} = \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k >, \tag{6}$$

where $\mathbf{x}'_{im}$ denotes the m-th element of the vector $\mathbf{x}'_i$, $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_M\}$ contains the trainable parameters of the MLP with M output channels. For the operation defined above, the following theorem is derived.

Theorem 1 *Given that parameters $\boldsymbol{\theta}_m$ in the network follow an independent Gaussian distribution with 0 mean and σ^2 variance, the distance of two points in the input space is upper bounded by the neighborhood distance of the corresponding points in the output space up to a scaling factor, and lower bounded by the neighborhood centroid distance of the same points up to a scaling factor,* i.e.,

$$\begin{aligned} &\sigma^2 K^2 \| \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_i^k - \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_j^k \|_2^2 \\ &\leq \mathbb{E}[\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2] \leq \sigma^2 dKM \sum_{k=1}^{K} \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2. \end{aligned} \tag{7}$$

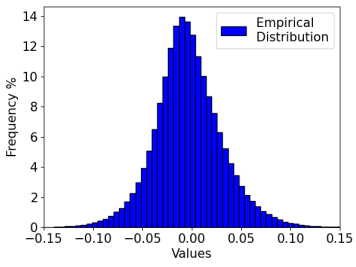


Figure 3: The empirical weight distribution of a layer in a fully trained dynamic GCN for point cloud classification. The distribution is Gaussian-like.

The condition in the theorem is reasonable since the parameters in neural networks are not only often initialized with independent Gaussian distributions. Actually, as shown in Fig. 3, also after training, the parameters tend to follow Gaussian-like empirical distribu-

Proof. **Upper bound.** The squared distance between two

points $\mathbf{x}'_i$ and $\mathbf{x}'_j$ after the graph convolution is

$$\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2 = \sum_{m=1}^{M}(\sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k - \mathbf{x}_j^k >)^2 \quad (8)$$

$$\leq \sum_{m=1}^{M} K \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k - \mathbf{x}_j^k) >^2 \quad (9)$$

$$\leq K \sum_{m=1}^{M}\sum_{k=1}^{K} \|\boldsymbol{\theta}_m\|_2^2 \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2. \quad (10)$$

The inequality in Eqn. 9 implies that the arithmetic mean is not larger than the quadratic mean while the inequality in Eqn. 14 follows from the Cauchy–Schwarz inequality. Assume that the parameters $\boldsymbol{\theta}_m$ in the network are random variables that follow a Gaussian distribution with 0 mean and σ^2 variance. Then the distance $\|\mathbf{x}'_i - \mathbf{x}'_k\|_2^2$ is also a random variable and the expectation is expressed as,

$$\mathbb{E}[\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2] \leq \mathbb{E}[K \sum_{m=1}^{M}\sum_{k=1}^{K} \|\boldsymbol{\theta}_m\|_2^2 \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2] \quad (11)$$

$$= \sigma^2 dKM \sum_{k=1}^{K} \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2. \quad (12)$$

Lower bound. Let $\mathbf{a}_m = \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k - \mathbf{x}_j^k >$. Then the squared distance in Eqn. 8 becomes

$$\sum_{m=1}^{M}(\sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k - \mathbf{x}_j^k >)^2 = \sum_{m=1}^{M} \mathbf{a}_m^2. \quad (13)$$

Using the Cauchy-Schwarz inequality

$$\sum_{m=1}^{M} \mathbf{a}_m \mathbf{b}_m \leq \sqrt{\sum_{m=1}^{M} \mathbf{a}_m^2} \sqrt{\sum_{m=1}^{M} \mathbf{b}_m^2} \quad (14)$$

and letting $\mathbf{b}_m^2 = 1/M$, then Eqn. 14 becomes

$$\left(\frac{1}{\sqrt{M}} \sum_{m=1}^{M} \mathbf{a}_m\right)^2 \leq \sum_{m=1}^{M} \mathbf{a}_m^2. \quad (15)$$

Thus, the lower bound of Eqn. 8 becomes

$$\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2 \geq \frac{1}{M} < \sum_{m=1}^{M} \boldsymbol{\theta}_m, \sum_{k=1}^{K} \mathbf{x}_i^k - \mathbf{x}_j^k >^2 . \quad (16)$$

Let $\boldsymbol{\phi} = \sum_{m=1}^{M} \boldsymbol{\theta}_m$ and $\mathbf{z} = \sum_{k=1}^{K} \mathbf{x}_i^k - \mathbf{x}_j^k$. Then

$$\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2 \geq \frac{1}{M}(\sum_{l=1}^{d} \boldsymbol{\phi}_l, \mathbf{z}_l)^2 = \frac{1}{M} \sum_{l=1}^{d}\sum_{n=1}^{d} \boldsymbol{\phi}_l \boldsymbol{\phi}_n \mathbf{z}_l \mathbf{z}_n. \quad (17)$$

Taking the expectation on both sides, Eqn. 17 becomes

$$\mathbb{E}[\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2] \geq \frac{1}{M} \sum_{l=1}^{d}\sum_{n=1}^{d} \mathbb{E}[\boldsymbol{\phi}_l \boldsymbol{\phi}_n] \mathbf{z}_l \mathbf{z}_n. \quad (18)$$

The elements of $\boldsymbol{\theta}_m$ follow an independent Gaussian distribution with 0 mean and σ^2 variance and $\boldsymbol{\phi} = \sum_{m=1}^{M} \boldsymbol{\theta}_m$. Then the elements of $\boldsymbol{\phi}$ follow an independent Gaussian distribution with 0 mean and $M\sigma^2$ variance. Thus,

$$\mathbb{E}[\boldsymbol{\phi}_l \boldsymbol{\phi}_n] = \begin{cases} 0 & l \neq n \\ M\sigma^2 & l = n \end{cases}. \quad (19)$$

Substituting Eqn. 19 into Eqn. 18, the lower bound becomes

$$\mathbb{E}[\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2] \geq = \sigma^2 K^2 \|\frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_i^k - \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_j^k\|_2^2. \quad (20)$$

□

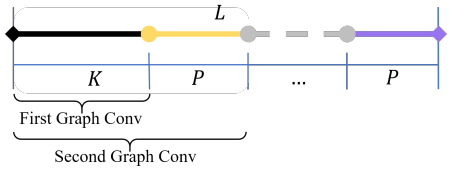


Figure 4: Neighbor sampling for MLPs sharing the same KNN search.

Since both the neighborhood distance and the neighborhood centroid distance are an indicator of the closeness of two points, Theorem 1 indicates that the adjacency property of points propagates smoothly across the stack of multilayer graph convolutions. This conclusion motivates us to rethink the frequently occurring adjacency assessment of points via KNN in a multilayer GCN. One straightforward bypass is to reduce the number of KNN searches and let several graph convolution modules share the results from one KNN search as shown. Yet, this simple scheme could probably reduce the receptive field for the stack of several graph convolutions.

Thus, we propose to progressively enlarge the receptive field of the graph convolutions that share the same KNN search as shown in Fig 4. As shown in Fig. 2b, for a stack of n graph convolutions, only one full-fledged KNN search is conducted, which leads to an enlarged pool of $L = K + (n-1)P$ neighbors more than originally needed. Then for the first convolution, the first K-nearest neighbors from the pool are still selected. For the l-th convolutions, the neighbors of a point are identified by randomly sampling the first $K + (l-1)P$ elements of the pool. That is, for each additional graph convolution, the sampling pool is enlarged by a step P. Since the bound in Theorem 1 is not strict, the adjacency between point could still change across

the deep layers. This allows the GCN to still be able to capture long-range dependencies between points after the computational simplification (See Fig. 6). The above operations reduce multiple KNN searches to one. Across the whole network, KNN search is conducted every several layers. This simplification can accelerate the inference of the network.

4.3. Graph convolution with graph feature gathering

The graph convolution in Eqn. 3 is applied to the subgraph defined by local point proximity. Subgraph features are gathered from the neighbors of a point and brought to the center of the local coordinate system. Then an MLP is applied to the centered features. In summary, the convolution in Eqn. 3 is of the following form

$$g(\mathcal{G}_i; \mathbf{\Theta}) = \sum_k f(\mathbf{x}_k - \mathbf{x}_i, \mathbf{\Theta}_k). \tag{21}$$

This form of operation is used in a couple of dynamic [49, 54] and non-dynamic [36, 23, 14] GCNs. To conduct the operation, features are first gathered from the neighborhood, *i.e.*, $\mathbf{x}_k - \mathbf{x}_i$, which forms a tensor with dimension $N \times K \times d$. Then the gathered feature is convolved with the MLP. The computational complexity of the convolution operation is $dKMN$. To save computations, we propose to shuffle the order of graph feature gathering and MLP. That is, the computation is conducted as $\sum_k f(\mathbf{x}_k) - f(\mathbf{x}_i)$. In this way, the MLP is first applied to the individual points, after which feature gathering is applied.

To explain the rationale of this shuffling operation, we consider a case widely used in GCNs [49], *i.e.*

$$g(\mathcal{G}_i) = \max_k \{(< [\boldsymbol{\theta}_m, \boldsymbol{\phi}_m], [\mathbf{x}_k - \mathbf{x}_i, \mathbf{x}_i] >)\}, \tag{22}$$

where $\max$ is used as the aggregation function and the operator $[\cdot]$ concatenates two vectors. It is claimed that the term $\mathbf{x}_k - \mathbf{x}_i$ captures the local information while $\mathbf{x}_i$ keeps the global information. The following theorem states that an equivalent but computationally efficient procedure exists for the special case in Eqn. 22.

Theorem 2 *For the graph convolution defined by Eqn. 22, shuffling the order of neighbor feature gathering and the MLP leads to an* ***equivalent*** *operation for the GCN.*

Proof. Eqn. 22 could be written as

$$\mathbf{x}'_{im} = \max_k \{(< \boldsymbol{\theta}_m, \mathbf{x}_k > + < \boldsymbol{\psi}_m, \mathbf{x}_i >)\}, \tag{23}$$

where $\boldsymbol{\psi}_m = \boldsymbol{\phi}_m - \boldsymbol{\theta}_m$. In Eqn. 23, the operations are rearranged such that the convolutions w.r.t. the points and their neighbors are perfectly separated. Considering that a point could act as a neighbor of the other point multiple times and the same parameters $\boldsymbol{\theta}_m$ are used for convolution, the convolution operation in Eqn. 23 has an equivalent procedure: 1) applying two different MLPs to the original points, 2) gathering the neighbor features, and 3) summing up the features. The computational complexity is reduced to $2dMN$, which is only $1/K$ of the original. □

Thus, inspired by the equivalent operation for Eqn. 23 obtained by shuffling the order of graph feature gathering and MLP, we propose to use the same shuffling procedure for the general case in Eqn. 22. The effectiveness of the shuffling operation is validated in the experiments.

5. Experiments

This section validates the effectiveness of the proposed GCN acceleration method on the four popular network architectures DGCNN [49], PointCNN [23], Point2Mesh [14], and [54]. Experiments on four important tasks are included, *i.e.*, point cloud classification, part segmentation, semantic segmentation, and surface reconstruction. For classification and segmentation, we evaluate the performance on the public benchmarks ModelNet40 [50], ShapeNetPart [3], and S3DIS [1]. For surface reconstruction, we use the dataset released by [14] and some public 3D models. All of the experiments are rerun for the original and the accelerated networks. The same training protocol is used for fair comparison. For point cloud classification, semantic segmentation, and part segmentation, the accuracy results are averaged over 5 runs, thus increasing the reliability of the reported numbers. The aim of our experiment is to compare the accuracy, test time, maximum GPU memory of the proposed method with original networks. The training of our accelerated models are all done on a single TITAN XP GPU whereas the original networks require more than one GPU for some of the experiments. For the detailed settings of each experiment, please refer to the supplementary material. Due to the different hardware environments, the runtime might be different from the original papers.

Hyperparameter Setup. Several hyperparameters are involved. We follow the default settings to determine the number of neighbors K in KNN. For classification with 1024 and 2048 points, K is 20 and 40, resp. For part segmentation and semantic segmentation, K is set to 40 and 20, resp. The enlargement step P is chosen empirically for different tasks. We try $P = 1/4K, 1/2K, 3/4K, K$.

Point Cloud Classification. The comparison between the original networks [49, 23, 54] and the accelerated versions for point cloud classification is shown in Table 1. With 1024 points available, compared with DGCNN, the accelerated network is about twice faster, reduces the GPU memory and computation by 49% and 76.6% with similar accuracy. On the heavier network [54], the accelerated version is about $\times 4$ times faster. Even for the compact PointCNN,

Network	Method	Points	K	OV Acc.	BL Acc.	Time [ms]	Mem. [GB]	FLOPs [G]	#GPU
PointNet [35]		1024	20	89.4	83.7	4.7	0.5	–	1
PointNet++ [36]		1024	20	90.7	–	113	–	–	1
KPConv [45]		1024	20	92.9	–	108	3.2	–	1
PointCNN [23]	Baseline	1024	20	91.86	87.93	35.3 / 100.%	0.8 / 100.%	2.5 / 100.%	1
	Accel.	1024	20	91.86	87.92	29.1 / 82.4 %	0.6 / 76.7 %	1.9 / 76.2 %	1
DGCNN [49]	Baseline	1024	20	92.10	89.05	74.7 / 100.%	5.2 / 100.%	86.0 / 100.%	1
	Accel.	1024	20	92.56	89.62	38.1 / 51.0%	2.5 / 48.1%	20.4 / 23.7%	1
[54]	Baseline	1024	20	92.54	89.57	95.4 / 100.%	4.9 / 100.%	74.2 / 100.%	1
	Accel.	1024	20	92.50	89.38	24.1 / 25.3%	1.3 / 26.5%	13.5 / 18.3%	1
DGCNN [49]	Baseline	2048	40	92.56	89.90	385.8 / 100.%	20.5 / 100.%	309.2 / 100.%	3
	Accel. S1	2048	40	92.58	89.60	212.7 / 55.1%	20.5 / 100.%	274.8 / 88.9%	3
	Accel. S2	2048	40	92.63	90.16	164.7 / 42.7%	8.7 / 42.4%	75.4 / 24.4%	1
	Accel.	2048	40	92.63	89.82	132.0 / 34.2%	8.8 / 42.9%	41.0 / 13.3%	1

Table 1: Quantitative comparison for point cloud classification on ModelNet40. All experiments are rerun and the accuracy results are averaged over 5 runs. OV Acc. and BL Acc. denote overall and balanced accuracy, resp.

Network	Method	Points	mIoU	Runtime [ms]	GPU mem. [GB]	FLOPs [G]	#GPU
PointCNN [23]	Baseline	2048	83.34	123.0 / 100.%	3.3 / 100.%	9.7 / 100.%	1
	Accel.	2048	83.21	111.9 / 91.0%	2.7 / 82.7%	7.6 / 78.8%	1
DGCNN [49]	Baseline	2048	84.95	116.1 / 100.%	17.2 / 100.%	158.8 / 100.%	2
	Accel.	2048	84.78	81.8 / 70.5%	4.1 / 23.8%	71.2 / 44.8%	1
[54]	Baseline	2048	84.13	365.3 / 100.%	9.7/ 100.%	202.9 / 100.%	2
	Accel.	2048	84.02	46.2 / 12.6 %	1.9 / 19.5%	52.7 / 26.0%	1

Table 2: Quantitative comparison for part segmentation of point clouds in ShapeNetPart. Experiments are rerun. Accuracy is averaged over 5 runs.

Figure 5: Surface reconstruction results. (a) Input point cloud. (b, c) Surface and normal map reconstructed by Point2Mesh. (d, e) Surface and normal map reconstructed by our method.

Method	F-score		Runtime [s]	#Param. [k]
	Bunny	Bird		
Baseline	69.7	53.3	0.41 / 100.%	735.8 / 100.%
Accel.	73.0	51.6	0.29 / 70.7%	153.7 / 20.9%

Table 3: Quantitative comparison for surface reconstruction.

Method	mIoU	Runtime	GPU mem.	#GPU
Baseline	57.5	172.7 / 100.%	14.6 / 100.%	2
Accel.	57.0	87.0 / 50.4%	6.0 / 40.8%	1

Table 4: Comparison for semantic segmentation of point clouds in S3DIS. Accuracy reported over 5 runs. The unit of the metrics is the same as that in Table 1.

the proposed method could reduce the runtime by 17.6%. When 2048 points are available, the accelerated version is about $\times 3$ faster, reduces GPU memory by 57.1% and computation by 86.7% without loss of accuracy. DGCNN needs

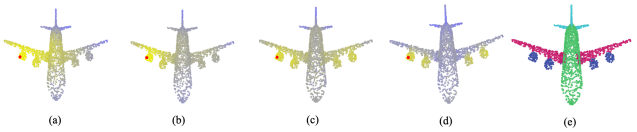


Figure 6: Visualization of the point distance across the accelerated network for part segmentation task. The distance of points to the red point in the figures is computed. Lighter color means closer distance. (a) Distance between points in the raw data. (b)-(d) Distance between point in the feature space from Layer 1, Layer 2, Layer 3 of the accelerated network. (e) Segmentation result. The accelerated network could still capture long-range dependency between points (*e.g.*wings of the plane).

three Titan Xp GPUs for the test with 2048 points, while our method only needs one GPU. Besides our full method, performances with only KNN simplification (Accel. S1) or operation shuffling in Sec. 4.3 (Accel. S2) are also ablated. As shown in the table, both of the strategies could improve the efficiency of the network. The point sampling based methods like Grid-GCN [51] and RandLA-Net [19] can speed up the network inference, while this direction is parallel to our method.

The reduction of inference time comes from the reduction of computation. The reduction of GPU memory mainly comes from shuffling of operations because it avoids the expansion of point feature. The effects of the proposed two methods depend on the redundancy of the baseline networks. For example, PointCNN exploits point subsampling to make the computation highly efficient while DGCNN and link DGCNN keep a high density of points across the network. Applying the same method to a more redundant network could bring more benefits.

Point Cloud Part Segmentation. The experimental results for point cloud part segmentation are shown in Table 2. The mean IoU metric is used to quantitatively evaluate the segmentation performance. As shown in Table 2, compared with DGCNN, our method greatly reduces the runtime, GPU memory consumption, and computation by 29.5%, 76.2%, and 56.2%, resp. The networks in [23] and [54] are accelerated by 9% and 77.4%, resp. In Fig. 6, the distance of points in the input space and the feature space is shown. As the network gets deeper, the accelerated network could still learn the long-range dependencies between points. As for the semantic segmentation task, Table 4 shows that our method reduces the runtime and memory consumption by 49.6% and 59.2% compared with DGCNN.

Surface Reconstruction. In order to validate the efficiency of our method applied to prior networks, we compare it with Point2Mesh [14] for the surface reconstruction. Similar to Point2Mesh, we use the F-score as the metric to evaluate the quality of the reconstructed meshes. The result

Model	Method	Classification	Part segmentation
		Conv / KNN/ Total	Conv / KNN / Total
PointCNN	Baseline	2.4G / 0.118G / 2.5G	8.5G / 1.175G / 9.7G
	Accel.	1.8G / 0.104G / 1.9G	7.0G / 0.588G / 7.6G
DGCNN	Baseline	77.3G / 8.7G / 86.0G	140.8G / 18.0G / 158.8G
	Accel.	20.3G / 0.1G / 20.4G	53.6G / 17.6G / 71.2G
Linked DGCNN	Baseline	60.9G / 13.3G / 74.2G	149.7G / 53.2G / 202.9G
	Accel.	13.4G / 0.10G / 13.5G	52.3G / 0.4G / 52.7G

Table 5: Breakdown analysis of FLOPs of different networks.

is shown in Table 3. It can be observed that our method has similar reconstruction quality as Point2Mesh, while speeding up inference by 29%. Note that the number of parameters is reduced by 79.1%. The qualitative results for different shapes are shown in Fig. 5. It is obvious that our accelerated method recovers the 3D meshes with a similar quality as Point2Mesh.

Applicability. The two methods proposed in this paper have different application scenarios. **I.** In the first method, we handle the simplification of neighbor querying. We focus on GCNs with KNN because they generally performs better and KNN is the most expensive method among the available methods. In the analysis of Theorem 1, the core assumption is the ordered neighbors. It does not matter whether the neighbors come from KNN search or other alternatives such as ball querying. Thus, the same theoretical analysis and conclusion hold for those methods. **II.** The second method is not limited to a specific GCN. Instead, it is applicable to any GCN with the computation pattern that feature gathering occurs before MLP (*e.g.*Point2Mesh). Actually, the second method is adopted for all the four investigated networks. In Table 5, the reduction of the FLOPs of convolution reflects how the shuffling trick works for different networks. **III.** Some classical GCNs do not use KNN. Yet, they still need to propagate message between neighbors. In this case (GCN2Conv, RGCNConv, Point2Mesh), the adjacency is usually defined on the input data. In some cases (MeshConv, Point2Mesh), feature gathering occurs before MLP and our shuffling method could be used.

6. Conclusion

In this work we have presented two strategies for improving the time and memory efficiency of dynamic GCNs. The two strategies are based on the analysis of basic operations in GCNs. The modified networks retain their accuracy while significantly shrinking the test time and GPU memory consumption. Experimental results show that our method has a significant performance on multiple important tasks. In the future, we plan to explore how to add flexibility and efficiency to the design of neural network for 3D tasks.

Acknowledgments: This work was partly supported by the ETH Zürich Fund (OK) and an Amazon AWS grant.

References

[1] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1534–1543, 2016. 6

[2] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *Advances in neural information processing systems*, pages 4868–4879, 2019. 1

[3] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 1, 6

[4] He Chen and Gregory S Chirikjian. Curvature: A signature for action recognition in video sequences. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 858–859, 2020. 3

[5] He Chen, Pengfei Guo, Pengfei Li, Gim Hee Lee, and Gregory Chirikjian. Multi-person 3d pose estimation in crowded scenes based on multi-view geometry. In *European Conference on Computer Vision*, pages 541–557. Springer, 2020. 3

[6] 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. 3

[7] Pim de Haan, Taco Cohen, and Max Welling. Natural graph networks. *arXiv preprint arXiv:2007.08349*, 2020. 3

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016. 1

[9] Hongyang Gao and Shuiwang Ji. Graph U-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019. 3

[10] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 15580–15592, 2019. 1

[11] Shuhang Gu, Yawei Li, Luc Van Gool, and Radu Timofte. Self-guided network for fast image denoising. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2511–2520, 2019. 3

[12] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. Pct: Point cloud transformer. *arXiv preprint arXiv:2012.09688*, 2020. 3

[13] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proc. ICLR*, 2015. 3

[14] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Point2mesh: A self-prior for deformable meshes. *arXiv preprint arXiv:2005.11084*, 2020. 1, 2, 3, 6, 8

[15] 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. 3

[16] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proc. ICCV*, pages 1389–1397, 2017. 3

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 3

[18] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 3

[19] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. RandLA-Net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11108–11117, 2020. 1, 3, 8

[20] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 3

[21] Chiyu Jiang, Jingwei Huang, Andrea Tagliasacchi, Leonidas Guibas, et al. Shapeflow: Learnable deformations among 3d shapes. *arXiv preprint arXiv:2006.07982*, 2020. 3

[22] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pages 3734–3743. PMLR, 2019. 3

[23] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on χ-transformed points. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 828–838, 2018. 1, 2, 6, 7, 8

[24] Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: A non-uniform discretization for neural networks. *arXiv preprint arXiv:1909.13144*, 2019. 3

[25] Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proc. CVPR*, 2020. 3

[26] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *Proc. ICCV*, pages 5623–5632, 2019. 3

[27] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. DHP: Differentiable meta pruning via hypernetworks. In *Proc. ECCV*, pages 608–624. Springer, 2020. 3

[28] Yawei Li, Wen Li, Martin Danelljan, Kai Zhang, Shuhang Gu, Luc Van Gool, and Radu Timofte. The heterogeneity hypothesis: Finding layer-wise differentiated network architectures. In *Proc. CVPR*, pages 2144–2153, 2021. 3

[29] Yangyan Li, Soeren Pirk, Hao Su, Charles R Qi, and Leonidas J Guibas. Fpnn: Field probing neural networks for 3d data. In *Advances in Neural Information Processing Systems*, pages 307–315, 2016. 3

[30] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. MetaPruning: Meta learning for automatic neural network channel pruning. In *Proc. ICCV*, 2019. 3

[31] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *Proc. ICLR*, 2019. 3

[32] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018. 3

[33] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 3

[34] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. *arXiv preprint arXiv:2003.04618*, 2020. 3

[35] 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, 3, 7

[36] 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, 3, 6, 7

[37] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020. 1

[38] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Pytorch3d, 2020. 1

[39] Davis Rempe, Tolga Birdal, Yongheng Zhao, Zan Gojcic, Srinath Sridhar, and Leonidas J Guibas. Caspr: Learning canonical spatiotemporal point cloud representations. *Advances in Neural Information Processing Systems*, 33, 2020. 3

[40] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 3

[41] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 3

[42] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10529–10538, 2020. 3

[43] Weijing Shi and Raj Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1711–1719, 2020. 1

[44] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 3

[45] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6411–6420, 2019. 7

[46] Frederick Tung and Greg Mori. Similarity-preserving knowledge distillation. In *Proc. CVPR*, pages 1365–1374, 2019. 3

[47] 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. 3

[48] 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. 1

[49] 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):1–12, 2019. 1, 2, 3, 6, 7

[50] 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. 6

[51] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. Grid-gcn for fast and scalable point cloud learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5661–5670, 2020. 3, 8

[52] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018. 3

[53] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. In *Advances in Neural Information Processing Systems*, pages 11983–11993, 2019. 1

[54] 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. 1, 2, 6, 7, 8

[55] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE transactions on image processing*, 26(7):3142–3155, 2017. 3

[56] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018. 3

[57] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE TPAMI*, 38(10):1943–1955, 2015. 3

[58] Zhiyuan Zhang, Binh-Son Hua, and Sai-Kit Yeung. Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1607–1616, 2019. 1

[59] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point transformer. *arXiv preprint arXiv:2012.09164*, 2020. 3

[60] Yongheng Zhao, Tolga Birdal, Jan Eric Lenssen, Emanuele Menegatti, Leonidas Guibas, and Federico Tombari. Quaternion equivariant capsule networks for 3d point clouds. *arXiv preprint arXiv:1912.12098*, 2019. 3

[61] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016. 3