THE CARD SHUFFLING HYPOTHESES: BUILDING A TIME AND MEMORY EFFICIENT GRAPH CONVOLUTIONAL NETWORK

Anonymous authors
Paper under double-blind review

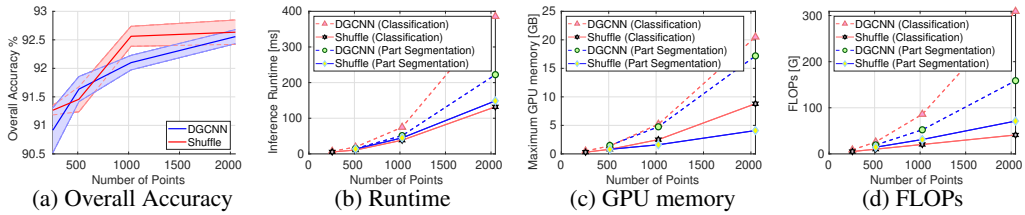
ABSTRACT

This paper investigates the design of time and memory efficient graph convolutional networks (GCNs). State-of-the-art GCNs adopt K-nearest neighbor (KNN) searches for local feature aggregation and feature extraction operations from layer to layer. Based on the mathematical analysis of existing graph convolution operations, we articulate the following two *card shuffling hypotheses*. (1) Shuffling the nearest neighbor selection for KNN searches in a multi-layered GCN approximately preserves the local geometric structures of 3D representations. (2) Shuffling the order of local feature aggregation and feature extraction leads to equivalent or similar composite operations for GCNs. The two hypotheses shed light on two possible directions of accelerating modern GCNs. That is, reasonable *shuffling* of the *"cards"* (neighbor selection or local feature operations) can significantly improve time and memory efficiency. A series of experiments show that the network architectures designed based on the proposed *card shuffling hypotheses* decrease both the time and memory consumption significantly (*e.g.*, about 50% for point cloud classification and semantic segmentation), while maintaining comparable accuracy, on several important tasks in 3D deep learning, *i.e.*, 3D classification, part segmentation, semantic segmentation, and mesh generation.

1 INTRODUCTION

Recent years have seen a surge of interest in applying graph operations directly on 3D representations (Zhao et al., 2019; Li et al., 2019b; Chen et al., 2020; Nash et al., 2020) such as point cloud (Qi et al., 2017a;b) and triangle mesh (Deng et al., 2019; Gkioxari et al., 2019; Schult et al., 2020). Although graph convolutional networks (GCNs) (Defferrard et al., 2016; Chami et al., 2019; Gasse et al., 2019; Yun et al., 2019) leverage geometrical and topological properties in each layer (Xu et al., 2018; Hanocka et al., 2019; Wang et al., 2019; Hanocka et al., 2020), it can be quite computationally expensive in both runtime and memory consumption. Especially, as the number of points in a 3D point cloud grows, runtime and memory consumption increase quite fast as shown in Figure 1, making the training process very slow or impossible with a single graphic card. Yet, most research on GCN ignores these problems and and resorts to multiple powerful GPUs.

GCNs are composed of a stack of graph convolutions which in turn contains a KNN search step and a multi-layer perceptron (MLP). The KNN search returns the indices of the K points closest to the inquired point, which is used to select the appropriate local features. The MLP extracts higher level of features gradually with a stack of graph convolutions. It is exactly the two components that are responsible for the expensive computation in GCNs. On the one hand, KNN is frequently called in GCN and the computation cost grows quadratically with the increase of number of points. On the other hand, the MLP is applied to the aggregated local features of each vertex to propagate information along the edges. For the first component, improvements have been carried out in recent years either by approximating KNN through Morton encoding (Morton, 1966; Li et al., 2012; Schertler et al., 2017; Thabet et al., 2019) or by accelerating KNN by optimizing CUDA programming through template meta programming (Ravi et al., 2020b;a). Yet, for the second component, very few research is directed, *i.e.*, optimizing graph convolution procedure. In this paper, we argue that both of the two computation-intensive operations can be simplified and accelerated, which could reduce the time and memory consumption of GCNs tremendously.



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

Figure 1: Comparison between DGCNN and the proposed method (Shuffle) on point cloud classification and part segmentation. (a) Overall accuracy for classification. Mean and Variance reported for 5 runs. (b) Runtime, (c) GPU memory consumption, and (d) FLOPs are reported for comparison. The proposed method can achieve significant reduction of computation resources without accuracy drop.

With the mathematical analysis, we find that the distance between two points after graph convolution is upper bounded by the neighborhood distance and lower bounded by the neighborhood centroid distance between the corresponding points in the input Euclidean space. That is, the local geometric structure is preserved after the graph convolution. Thus, the practice of repeated calling the computation-intensive KNN search for the graph convolution in a multi-layered GCN is worth pondering. Instead, the KNN searches except the first one could be accelerated by shuffling the nearest neighbor selection criterion based on the search results of the first one. When retrieving local features, instead of calculating the pairwise distances multiple times on high-dimensional feature space, we can calculate that only once on the 3D input Euclidean space and preserve the indices of more neighbors. For the latter usage, the indices are shuffled and randomly sampled to form the final K-nearest neighbors. So we articulate the first hypothesis:

The Card Shuffling Hypothesis 1. *Shuffling the nearest neighbor selection for KNN searches in a multi-layered GCN approximately preserves the local geometric structures of 3D representations.*

On the other hand, the computation procedure of graph convolution is also revisited in this paper. In the previous graph convolution, the neighbor features are first gathered and then the convolution operation is conducted. The feature gathering results in an expanded tensor with repeated entries which accounts for the tremendous increased computation resources for the following convolution. Starting from the basic EdgeConv, we find that a mathematically equivalent but computation-efficient procedure could be derived. The key idea is shuffling the orders of neighbor feature gathering and convolution operation. In this way, the convolution is conducted merely on the non-expanded feature tensors. For the simple case in EdgeConv, this shuffling will not deteriorate performance. For the complex cases in EdgeConv and MeshConv, we generalize the core idea by the second hypothesis:

The Card Shuffling Hypothesis 2. *Shuffling the order of local feature aggregation and feature extraction leads to equivalent or similar composite operations for GCNs. Reasonable arrangement of the order significantly improves time and memory efficiency.*

The contributions of this paper can be summarized as: **I.** We propose *the shuffling cards hypotheses* as a new perspective on designing fast and memory efficient graph convolutional networks. **II.** Guided by the shuffling cards hypotheses, we propose two directions to accelerate modern GCNs for 3D point cloud processing. **III.** The GCNs with our novel design maintain comparable accuracy while decreasing both the runtime and memory consumption significantly. All the training can be completed on a single TITAN XP graphics card.

2 Notations and Preliminaries

In order to formally define the acceleration problem for GCNs, we begin with a formal model built from important elements defined below in the context of this paper.

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

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

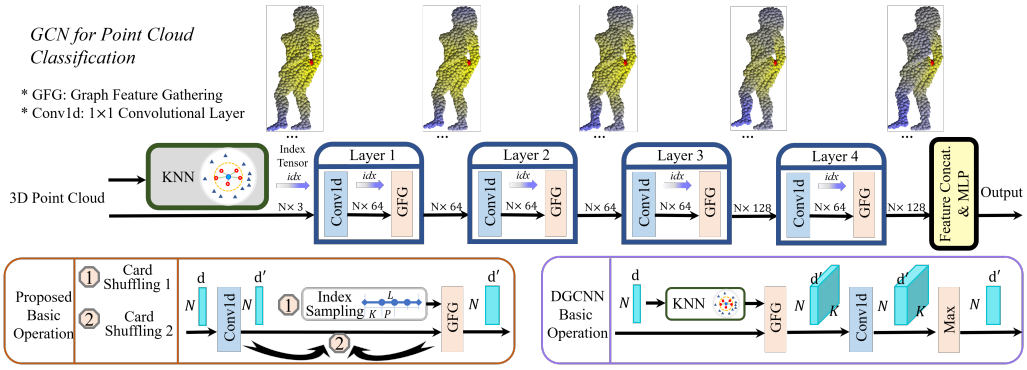


Figure 2: Overview of our pipeline and visualization of the feature space. The network architecture for all tasks basically follows MLP structure. The visualization of feature space above the pipeline is a rendering of distance color map extracted from each layer of the neural network on the shape of raw data. The two bottom sub-figures compare the basic operation of DGCNN and our method.

The neighbors of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is sorted according to the distance to them. To some extent, the neighborhood distance reflects the distance between two points. That is, two points with smaller neighborhood distance are highly likely to be closer compared to those with larger neighborhood distance.

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

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

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

Similar to neighborhood distance, neighborhood centroid distance is also a metric that reflects the closeness of two points in the Euclidean space.

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 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. The points on the point clouds and meshes serves as the vertices $\mathcal{V}$. The edges $\mathcal{E}$ can be either predefined as in a mesh or dynamically computed for a point cloud by algorithms such as KNN search. A subgraph $\mathcal{G}_i$ captures the local connectivity and are constructed slightly differently for point clouds and meshes.

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 $\mathcal{G}$. Formally, graph convolution can be defined in terms of the subgraphs,* i.e.*,*

$$\boldsymbol{f}_i = g(\mathcal{G}_i; \boldsymbol{\Theta}) = \square_{\boldsymbol{e}_i^k \in \mathcal{E}_i} h(\boldsymbol{e}_i^k; \boldsymbol{\Theta}_k), \tag{3}$$

where $\boldsymbol{e}_i \in \mathbb{R}^d$ is a vector that encodes the edge feature, $\boldsymbol{e}_i^k$ denotes an edge of the subgraph $\boldsymbol{e}_i^k \in \mathcal{E}_i$, $\boldsymbol{f}_i \in \mathbb{R}^M$ is the newly computed feature, $\boldsymbol{\Theta}_k \in \mathbb{R}^{d \times M}$ is the trainable parameters and is the same for all of the subgraphs $\mathcal{G}_i$, $\boldsymbol{\Theta} = \{\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2, \cdots, \boldsymbol{\Theta}_K\}$ is the ensemble of the trainable parameters.

The function $h(\cdot)$ is *the edge function* that transforms a d-dimensional input feature into a M-dimensional vector. It could have various forms in terms of the edge features. The edge function with trainable parameters $\boldsymbol{\Theta}_k$ is usually implemented as a MLP in existing works, which in turn is implemented as convolution operation. The operator $\square$ is *the aggregation function*. It is a symmetric function (*e.g.*, sum or max) that does not depend on the order of the edges. A stack of graph convolution and other operations such as pooling constitute a GCN.

In this paper, N represents the number of points, d represents the dimension of latent space features, K represents the number of neighbors for each point, and M represents number of dimension of the intermediate output channel. Without loss of clarity, the notations $\boldsymbol{x}$ and $\boldsymbol{e}$ are also used to denote the vertex and edge features.

3 SHUFFLING CARDS IN DYNAMIC GRAPH CNN

Dynamic Graph CNN (Wang et al., 2019) is commonly used for 3D point cloud processing because its local feature descriptor compensates the insufficiency of topological information that point cloud inherently suffers. However, as the number of points in 3D point cloud grows, training a DGCNN is quite memory consuming and the testing time became longer as well, prohibiting it from being applied to real world scenarios with larger scales. In this section, we assess the computational complexity in DGCNN, prove theorems that shed light on GCN acceleration, and describe the rationale of the proposed hypothesis.

EdgeConv (Wang et al., 2019) is a special form of the graph convolution in Eqn. 3 defined for point clouds which has the following form

$$\boldsymbol{x}_i' = \square_{k:(i,k)\in\mathcal{E}_k} h(\boldsymbol{x}_i, \boldsymbol{x}_i^k; \boldsymbol{\Theta}), \tag{4}$$

where $\{\boldsymbol{x}_i^k\}$ are the K-nearest neighbors of $\boldsymbol{x}_i$. The edge (i,k) connects $\boldsymbol{x}_i$ and it neighbor $\boldsymbol{x}_i^k$. The combination of the vertex features is used as the edge feature. Note that the same MLP is used for different edge features $\{\boldsymbol{x}_i, \boldsymbol{x}_i^k\}$ since the trainable parameters $\boldsymbol{\Theta}$ are the same. This constitutes a translation-invariant edge function. Note that, for EdgeConv, KNN search is first conducted to define the graph, which is followed by the convolution operation. In the following, the computational complexity of the two operations are analyzed and the acceleration methods are proposed.

3.1 ANALYSIS OF COMPUTATIONAL COMPLEXITY ON LATENT SPACE

Proposition 1 *The ratio of computational complexity between KNN operation and graph convolution operation is* $\gamma = \frac{N}{2KM}$.

Assume that the point cloud is represented by a $N \times d$ matrix $\boldsymbol{X}$. To compute the K-nearest neighbors of all of the points, pairwise comparison between the points is conducted, *i.e.*, $\boldsymbol{D} = \boldsymbol{X}\boldsymbol{X}^T$. 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 $\mathbf{X}$ with a dimension of $2d \times N \times K$. Then a 3D 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 part of EdgeConv.

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

$$\gamma = \frac{\mathcal{C}_{nn}}{\mathcal{C}_{conv}} = \frac{dN^2}{2dMKN} = \frac{N}{2KM} \tag{5}$$

Compared with the number of nearest neighbors K and the output channel M, the number of points in a point cloud could changes dramatically. When N is small, the computation occupied by the pairwise computation is relatively small and even negligible. But when the point cloud grows huge, the major computation could be consumed by the pairwise comparison.

3.2 POINT ADJACENCY PROPERTY BEFORE AND AFTER GRAPH CONVOLUTION

Graph convolution layers are stacked sequentially to form a dynamic GCN. The input 3D point cloud is transformed into the latent space by graph convolution layers. In the following, we investigate how local geometric structures propagate within the neural network 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. This results in a simplification and acceleration of the adjacency assessment in GCNs. For the simplicity of analysis, two assumptions are made.

Assumption 1 *Inner product and summation are selected as the edge function and the aggregation operation in Eqn. 4. That is, the EdgeConv in Eqn. 4 has the following form*

$$\boldsymbol{x}_i' = [\boldsymbol{x}_{i1}', \cdots, \boldsymbol{x}_{im}', \cdots, \boldsymbol{x}_{iM}'], \boldsymbol{x}_{im}' = \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \boldsymbol{x}_i^k >, \tag{6}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_M\}$ *is the trainable parameters of the MLP with* M *output channels.*

Assumption 2 *The parameters* $\boldsymbol{\theta}_m$ *in the network are random variables that follows independent Gaussian distribution with 0 mean and* σ^2 *variance.*

The EdgeConv in **Assumption 1** leads to a standard convolution operation. **Assumption 2** is reasonable since the parameters in GCNs are often initialized with independent normal distribution. Under the two assumptions, the following conclusion can be made.

Theorem 1 *Under Assumption 1 & 2, the distance of two points in the latent space is upper bounded by the neighborhood distance of the corresponding points in the Euclidean 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.*,*

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

The proof of the theorem is given in the Appendix B. Note that both the neighborhood distance and the neighborhood centroid distance are an indicator of the closeness of two points. Thus, Theorem 1 indicates that there should be no abrupt changes of the adjacency property between points where a an EdgeConv is applied. To a large extent, closer points in Euclidean space are also relatively closer to each other in the latent space. In other words, EdgeConv is a adjacency preserving operation.

The above conclusion motivates the rethinking of the frequently occurred adjacency assessment of the points via KNN in a multi-layered GCN. Instead of conducting the exact KNN search every time it is needed, one could simplify the latter KNN search based on the adjacency property obtained by the first one. This simplification is summarized in Algorithm 1 (Appendix C). For a GCN with n graph convolutions, only one full-fledged KNN search with the computation of pairwise distance is conducted, which keeps a pool of L ($L = K + (n-1)P$) neighbors more than originally needed. Then for the first EdgeConv, the K-nearest neighbors from the pool are still selected. For the l-th EdgeConvs, the neighbors of a point are identified by randomly sampling the first $K + (l-1)P$ elements of the pool. Note that the sampling range is progressively enlarged and P is the enlargement step. The random sampling actually shuffles the neighbor selection criterion. Since the local geometric structure is not strictly preserved by EdgeConvs, the adjacency between point might still change with the EdgeConvs to some extent. Random sampling provides the possibility of selecting different neighbors for the EdgeConvs. The above operations only involves one KNN search compared with the multiple searches in the original DGCNN. This simplification can accelerate the inference of the network.

The simplification of the KNN procedure is based on the conclusions from Theorem 1 which in turn is derived under two assumptions. In actual applications, the two assumptions might be violated and the validity of the simplification remains to be tested. The stated **Card Shuffling Hypothesis 1** is the generalization of the simple case and we try to validate it with experiments.

3.3 Edge Function in EdgeConv

The final EdgeConv used in DGCNN has the following form

$$\boldsymbol{x}_{im}' = \max_k \{\mathrm{ReLU}(< [\boldsymbol{\theta}_m, \boldsymbol{\phi}_m], [\boldsymbol{x}_k - \boldsymbol{x}_i, \boldsymbol{x}_i] >)\}, \tag{8}$$

where $\max$ is used as the aggregation function and the operator $[\cdot]$ concatenates two vectors. It is claimed that the term $\boldsymbol{x}_k - \boldsymbol{x}_i$ captures the local information while $\boldsymbol{x}_i$ keeps the global information. To conduct the operation defined above, one needs to first gather the edge features $[\boldsymbol{x}_k - \boldsymbol{x}_i, \boldsymbol{x}_i]$ for all of the point, which forms a tensor with dimension $2d \times N \times K$. Then the gathered feature is convolved with the MLP with the trainable parameters $[\boldsymbol{\theta}_m, \boldsymbol{\phi}_m]$. The computation complexity of the convolution operation is $2dKMN$. In the following theorem, we prove that it is possible to have an equivalent operation but with significantly reduced computational complexity.

Theorem 2 *For the EdgeConv defined by Eqn. 8, shuffling the order of neighbor feature gathering and the* 1×1 *convolution leads to* ***equivalent*** *operation for the GCN. Reasonable arrangement of the order significantly improves time and memory efficiency.*

Proof. Eqn. 8 could be written as

$$\boldsymbol{x}_{im}' = \max_k \{\mathrm{ReLU}(< \boldsymbol{\theta}_m, \boldsymbol{x}_k > + < \boldsymbol{\psi}_m, \boldsymbol{x}_i >)\}, \tag{9}$$

where $\boldsymbol{\psi}_m = \boldsymbol{\phi}_m - \boldsymbol{\theta}_m$. In Eqn. 9, the operations are rearranged such that the convolution wrt the points and their neighbors are perfectly separated. Considering that a point could acts as a neighbor of the other point for multiple times and the same parameters $\boldsymbol{\theta}_m$ are used for convolution, the

Table 1: Quantitative comparison for point cloud classification on ModelNet40.

Network	Overall Acc.	Balanced Acc.	Time [ms]	Mem. [GB]	FLOPs [G]	#GPU
1024 points, K=20						
DGCNN	92.10	89.05	74.7	5.2	86.0	1
Shuffle H1	92.58	89.56	53.3	5.2	77.4	1
Shuffle H2	92.89	90.30	54.8	2.5	29.0	1
Shuffle	92.56	89.62	38.1	2.5	20.4	1
2048 points, K=40						
DGCNN	92.56	89.90	385.8	20.5	309.2	3
Shuffle H1	92.58	89.60	212.7	20.5	274.8	3
Shuffle H2	92.63	90.16	164.7	8.7	75.4	1
Shuffle	92.63	89.82	132.0	8.8	41.0	1

convolution operation in Eqn. 9 has an equivalent procedure: 1) applying two different MLPs to the original points, 2) gathering the neighbor features, and 3) summing up the features. Thus, the order of neighbor feature gathering and convolution can be shuffled in the equivalent procedure. The computational complexity is reduced to $2dMN$, which is only $1/K$ of the original. The reduction of computation is due to the avoidance of the redundant computation for the repeated neighbors. □

The shuffling of the computation procedure is summarized in Algorithm 2 (Appendix C). An additional note is that in some implementations the edge function after the feature gathering procedure might be a composition of multiple convolution and batch normalization layers, *e.g.*, Conv-BN-ReLU-Conv-BN-ReLU. In this case, there is no exact equivalence like the one in Eqn. 9 for the complex edge functions. Thus, Theorem 2 does not absolutely hold. However, the shuffling operation could still reduce the computational complexity and memory consumption. In this case, we conjecture that the shuffled network could achieve similar performance as the original one. So we try to generalize the conclusion to more complex edge functions in **Card Shuffling Hypothesis 2** .

4 Shuffling Cards in Deep Prior Network

In this section, the second card shuffling operation, is extended to the Point2Mesh network for surface reconstruction (Hanocka et al., 2020). In order to reconstruct surface from point cloud, the network estimates differential vertex positions. Then, a particular vertex position is calculated by averaging the vertex prediction on edges connected to the vertex,

$$\boldsymbol{v}_i' = \frac{1}{n}\sum_{j\in n}\boldsymbol{e}_j', \boldsymbol{e}_j' = \sum_{k=1}^{5} < \boldsymbol{\Theta}_k, \boldsymbol{e}_j^k > \tag{10}$$

where $\boldsymbol{e}_j$ is the predicted position of the vertex $\boldsymbol{v}_i$ on edge $\boldsymbol{e}_j'$. In Eqn. 10, $\boldsymbol{e}_j$ is computed by MeshConv. Compared with the EdgeConv in Eqn. 4, the MLPs $\boldsymbol{\Theta}_k$ for each edge feature are different. This motivates us to use the same MLP $\boldsymbol{\Theta}$ for different edge features, *i.e.*,

$$\boldsymbol{e}_j' = \sum_{k=1}^{5} < \boldsymbol{\Theta}, \boldsymbol{e}_j^k >. \tag{11}$$

In this pipeline, similar process of gathering edge neighbor features and carrying out convolution on the repeated features exist. We optimize running time and GPU memory usage by shuffling the order of feature gathering and convolution according to **Card Shuffling Hypothesis 2**, *i.e.*, $\boldsymbol{e}_j' =< \boldsymbol{\Theta}, \sum_{k=1}^{5} \boldsymbol{e}_j^k >$. Thus, the computation procedure of the simplified and shuffled operation proceeds as follows: 1) gathering edge features; 2) conducting convolution with the MLP $\boldsymbol{\Theta}$.

5 Experiments

This section evaluates the proposed method according to *Shuffling Cards Hypotheses* on four important tasks, *i.e.*, point cloud classification, part segmentation, semantic segmentation, and surface reconstruction. For classification and segmentation, we evaluate performance on the public benchmarks ModelNet40 (Wu et al., 2015), ShapeNetPart (Chang et al., 2015), and S3DIS (Armeni et al., 2016). For surface reconstruction, we uses the dataset released by Hanocka et al. (2020) and some public 3D models. The aim of our experiment is to compare the accuracy, test time, maximum GPU

Table 2: Quantitative comparison for part segmentation of point cloud on ShapeNetPart.

Network	Mean IoU	Runtime [ms]	GPU mem. [GB]	FLOPs [G]	# Params. [M]	#GPU
DGCNN	84.95	116.1	17.2	158.8	1.5	2
Shuffle	84.50	72.3	4.09	70.9	1.5	1

Table 3: Comparison for semantic segmentation of point cloud on S3DIS

Network	Mean IoU	Overall Acc.	Runtime [ms]	GPU mem. [GB]	FLOPs [G]	#GPU
DGCNN	57.5	90.28	172.7	14.6	268.1	2
Shuffle	57.0	90.10	87.0	6.0	134.3	1

Table 4: Quantitative comparison for surface reconstruction.

Network	F-score Bunny	F-score Bird	Runtime [s]	GPU mem. [GB]	#Param. [k]
Point2Mesh	69.7	53.3	0.41	2.24	735.8
Shuffle	73.0	51.6	0.29	2.04	153.7

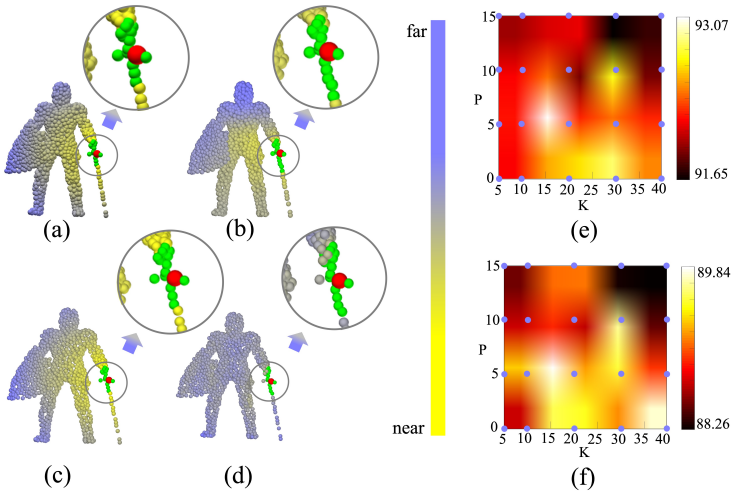


Figure 3: Qualitative result on Modelnet40. **Left:** (a), (b) Input space and last-layer feature space rendered as colormap between the red point and the rest points at epoch 0. The green points are KNN of the red point. (c), (d) follow the same layout with (a), (b) at epoch 250. **Right:** (e) Ablation study of overall acc. *w.r.t* parameters K and P. Values calculated are the points on the grid, and the hotmap is derived by bilinear interpolation. (f) follows the same layout with (e) for balanced acc.

memory of the proposed method with original networks. The training of our models are all done on single TITAN XP GPU. For detailed settings of each experiment, please refer to Appendix D.

Point Cloud Classification. In Table 1, besides comparing our full method, performance with hypothesis 1 only and hypothesis 2 only are also ablated. Detailed ablative study of how a wide range of K and P (defined in Sec. 3.2) affects performance is carried out in Figure 3 (e), (f). The evolving of feature space *w.r.t* number of epochs is shown in Figure 3 (a) - (d). The local structures are preserved and converge to smaller and smaller regions as the network propagates. More detailed feature space analysis can be found at Appendix E. Our method reduces the runtime and memory in all cases by approximately 50%. The FLOPs are also reduced by about 76%. DGCNN needs three Titan Xp GPUs for the test with 2048 points, while our method only needs one GPU.

Point Cloud Segmentation. The classifier can be adapted for part segmentation and semantic segmentation tasks by classifying each point in the point cloud into predefined part category labels. The experiments are run with the same setting as DGCNN. Besides classification accuracy, mean IoU metric is used to quantitatively evaluate the segmentation performance. As shown in Table 2, for part segmentation, our method greatly reduces the runtime and memory consumption by 38% and 76% respectively compared to DGCNN. As for semantic segmentation task, Table 3 shows that our method reduces the runtime and memory consumption by 50% and 59% respectively compared to DGCNN. Ablation study of the semantic segmentation performance *w.r.t.* parameters K and P in a wide range can be found at Figure 7 in Appendix F.

Figure 4: Surface reconstruction results. Both surface and normal map are visualized.

Surface Reconstruction. In order to validate efficiency of our method on deep prior network, we compare our method with Point2Mesh (Hanocka et al., 2020) for the surface reconstruction. Similar to Point2Mesh, we use the F-score as the metric to evaluate the quality of reconstructed meshes. The result is shown in Table 4. It can be observed that our method has similar reconstruction quality as Point2Mesh, while speeding up by 29%. To be noted that, the memory consumption is only reduced by 9% because the feature gathering operation still consumes the same memory in our accelerated version of Point2Mesh. The qualitative results of different shapes are shown in Figure 4 and Figure 8 of Appendix F. It's not difficult to notice that our accelerated method recovers the 3D meshes of comparable quality as Point2Mesh.

6 Related Work

Learning Based Methods for 3D Content. Deep neural networks have obtained success in 2D image recognition, but its extension to 3D is relatively unclear. Some methods operate directly on graph. In recent years, operations directly operated on graphs such as edge convolution (Wang et al., 2019), mesh convolution (Hanocka et al., 2020), graph pooling and unpooling (Hanocka et al., 2019) have been defined. But these networks are still based on basic network architectures such as MLP (Qi et al., 2017a;b; Wang et al., 2019), Unet (Schult et al., 2020), or Encoder-Decoder (Hanocka et al., 2020) and the network usually contains just a few layers instead of being very deep. Other methods operate on voxels (Choy et al., 2019; Mescheder et al., 2019; Niemeyer et al., 2019; Peng et al., 2020), so that 3D CNN for 2D images can be more easily adapted. From our perspective, all these methods are trying to represent the unordered data with uncertain dimension into fixed-dimensional features with the cost of flexibility for more efficient data structures such as octree. However, the memory consumption grows severely as either the resolution of grid or the number of layers of the network grows. This leads to our work, a new perspective that brings flexibility into the design of neural network architecture so that the time and memory efficiency can be significantly improved.

Acceleration and Compression of Neural Network. The computation and parameter efficiency of neural networks is a major concern for the deployment of them in resource-constrained requirements. There have been tremendous works that try to improve the efficiency of neural networks and accelerate their inference. The major techniques can be classified into network pruning (He et al., 2017; Liu et al., 2019c;a; Li et al., 2020), low-rank filter approximation (Zhang et al., 2015; Li et al., 2019a), network quantization (Han et al., 2015; Zhu et al., 2016; Li et al., 2019c), and knowledge distillation (Hinton et al., 2015). But most of the methods deal with networks for 2D images. A couple of more related works introduce pixel-voxel convolution for efficient 3D deep learning (Liu et al., 2019b; Tang et al., 2020). Different from those works, we try to accelerate GCNs by analyzing the basic operations including KNN and graph convolution which are computation-intensive.

7 Conclusion

In this work we have presented the cards shuffling hypotheses, a novel perspective of designing neural network architectures for 3D vision and graphics tasks which retains strong accuracy while significantly shrinking test time and GPU memory consumption. We have demonstrated the utility of our hypotheses into dynamic GCNN and self prior network. Experimental results show that our method has significant performance on multiple important tasks. In the future, we plan to explore how to give more flexibility and efficiency to the design of neural network for 3D tasks, for example, by introducing adaptive voxel sizes.

REFERENCES

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*, pp. 1534–1543, 2016.

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

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.

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. *arXiv preprint arXiv:2007.10986*, 2020.

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*, pp. 3075–3084, 2019.

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

Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. *arXiv preprint arXiv:1910.02370*, 2019.

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*, pp. 15580–15592, 2019.

Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9785–9795, 2019.

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.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Point2mesh: A self-prior for deformable meshes. *arXiv preprint arXiv:2005.11084*, 2020.

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

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

Shengren Li, Lance C Simons, Jagaseesh Bhaskar Pakaravoor, Fatemeh Abbasinejad, John D Owens, and Nina Amenta. kann on the gpu with shifted sorting. 2012.

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

Yawei Li, Vagia Tsiminaki, Radu Timofte, Marc Pollefeys, and Luc Van Gool. 3D appearance super-resolution with deep learning. In *Proc. CVPR*, 2019b.

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.

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

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*, 2019a.

Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. In *Advances in Neural Information Processing Systems*, pp. 965–975, 2019b.

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

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*, pp. 4460–4470, 2019.

Guy M Morton. A computer oriented geodetic data base and a new technique in file sequencing. 1966.

Charlie Nash, Yaroslav Ganin, SM Eslami, and Peter W Battaglia. Polygen: An autoregressive generative model of 3d meshes. *arXiv preprint arXiv:2002.10880*, 2020.

Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5379–5389, 2019.

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

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*, pp. 652–660, 2017a.

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*, pp. 5099–5108, 2017b.

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*, 2020a.

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

Nico Schertler, Marco Tarini, Wenzel Jakob, Misha Kazhdan, Stefan Gumhold, and Daniele Panozzo. Field-aligned online surface reconstruction. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

Jonas Schult, Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. Dualconvmesh-net: Joint geodesic and euclidean convolutions on 3d meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8612–8622, 2020.

Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. *arXiv preprint arXiv:2007.16100*, 2020.

Ali Thabet, Humam Alwassel, and Bernard Ghanem. Mortonnet: Self-supervised learning of local features in 3d point clouds. *arXiv preprint arXiv:1904.00230*, 2019.

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.

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*, pp. 1912–1920, 2015.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

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

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.

Rui Zhao, Kang Wang, Hui Su, and Qiang Ji. Bayesian graph convolution lstm for skeleton based action recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 6882–6892, 2019.

Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

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

A ADDITIONAL DEFINITIONS

The formal definitions of some other basic concepts including KNN, point cloud and mesh are given in this section.

Definition 5 (K-Nearest Neighbors) *Given a set of points $\mathcal{S} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ in d-dimensional latent space χ_d and a distance metric $\mathcal{D}$, the K-nearest neighbors of a point $\boldsymbol{x}_i$ is a subset $\mathcal{S}_i = \{\boldsymbol{x}_i^1, \boldsymbol{x}_i^2, \cdots, \boldsymbol{x}_i^K\} \subseteq \mathcal{S}$ that contains the K points closest to $\boldsymbol{x}_i$ w.r.t. the distance metric $\mathcal{D}$.*

Euclidean distance is usually selected as the distance metric. To conduct the KNN search, the pairwise distance $\mathcal{D}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ between every pair of points needs to be computed, which is quite computation-intensive. Thus, it is beneficial to accelerate or approximate KNN search.

Definition 6 (Point Cloud and Triangular Mesh) *A 3D point cloud is defined by a set of points $\mathcal{P} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ in 3D Euclidean space $\mathbb{R}^3$. A triangular mesh is a polygonal representation of a differentiable 2-manifold $\mathcal{M}^2$,* i.e.*, a strip of triangles that encapsulate the point cloud $\mathcal{P}$.*

The connectivity between points in a point cloud is not defined but can be dynamically computed via KNN search on the fly. For a mesh, the connectivity is defined by the triplet of vertices of the triangles. The connectivity is important for propagating information between the vertices. In this paper, the term *local geometric structure* is frequently used. It can be regarded as a descriptor that captures the local connectivity and shape of a subset of a point cloud or mesh.

B PROOFS

In this section, we provide the detailed proof of both the upper and lower bounds of Theorem 1 in Section 3.2.

B.1 PROOF OF THE UPPER BOUND IN THEOREM 1

Proof. **Upper bound.** For the simplicity of analysis, inner product and summation are selected as the edge function and the aggregation operation in Eqn. 4. Thus, the theorem is derived under the assumption that the EdgeConv in Eqn. 4 has the following form

$$\boldsymbol{x}_i' = [\boldsymbol{x}_{i1}', \cdots, \boldsymbol{x}_{im}', \cdots, \boldsymbol{x}_{iM}'], \tag{12}$$

$$\boldsymbol{x}_{im}' = \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \boldsymbol{x}_i^k >, \tag{13}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_M\}$ is the trainable parameters of the MLP with M output channels. Then the squared distance between two points $\boldsymbol{x}_i'$ and $\boldsymbol{x}_j'$ after the EdgeConv is

$$\|\boldsymbol{x}_i' - \boldsymbol{x}_j'\|_2^2 = \sum_{m=1}^{M} (\boldsymbol{x}_{im}' - \boldsymbol{x}_{jm}')^2 \tag{14}$$

$$= \sum_{m=1}^{M} (\sum_{k=1}^{K} < \boldsymbol{\theta}_m, \boldsymbol{x}_i^k > - \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \boldsymbol{x}_j^k >)^2 \tag{15}$$

$$= \sum_{m=1}^{M} (\sum_{k=1}^{K} < \boldsymbol{\theta}_m, \boldsymbol{x}_i^k - \boldsymbol{x}_j^k >)^2 \tag{16}$$

$$\leq \sum_{m=1}^{M} K \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \boldsymbol{x}_i^k - \boldsymbol{x}_j^k >^2 \tag{17}$$

$$\leq K \sum_{m=1}^{M} \sum_{k=1}^{K} \|\boldsymbol{\theta}_m\|_2^2 \|\boldsymbol{x}_i^k - \boldsymbol{x}_j^k\|_2^2. \tag{18}$$

The inequality in Eqn. 17 follows that the arithmetic mean is not larger than the quadratic mean while the inequality in Eqn. 24 follows Cauchy–Schwarz inequality. Assume that the parameters

$\boldsymbol{\theta}_m$ in the network are random variables that follows Gaussian distribution with 0 mean and σ^2 variance. Then the distance $\|\boldsymbol{x}'_i - \boldsymbol{x}'_k\|_2^2$ is also a random variable and the expectation is expressed as,

$$\mathbb{E}[\|\boldsymbol{x}'_i - \boldsymbol{x}'_j\|_2^2] \leq \mathbb{E}[K \sum_{m=1}^{M} \sum_{k=1}^{K} \|\boldsymbol{\theta}_m\|_2^2 \|\boldsymbol{x}_i^k - \boldsymbol{x}_j^k\|_2^2] \tag{19}$$

$$= K \sum_{m=1}^{M} \sum_{k=1}^{K} \mathbb{E}[\|\boldsymbol{\theta}_m\|_2^2] \|\boldsymbol{x}_i^k - \boldsymbol{x}_j^k\|_2^2 \tag{20}$$

$$= \sigma^2 dKM \sum_{k=1}^{K} \|\boldsymbol{x}_i^k - \boldsymbol{x}_j^k\|_2^2, \tag{21}$$

where the term $\sum_k \|\boldsymbol{x}_i^k - \boldsymbol{x}_j^k\|_2^2$ is just the neighborhood distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. □

B.2 Proof Of the Lower Bound in Theorem 1

Proof. **Lower bound.** In Eqn. 16, let

$$\boldsymbol{a}_m = \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \boldsymbol{x}_i^k - \boldsymbol{x}_j^k >. \tag{22}$$

Thus, Eqn. 16 become

$$\sum_{m=1}^{M} (\sum_{k=1}^{K} < \boldsymbol{\theta}_m, \boldsymbol{x}_i^k - \boldsymbol{x}_j^k >)^2 = \sum_{m=1}^{M} \boldsymbol{a}_m^2. \tag{23}$$

Using Cauchy-Schwarz inequality

$$\sum_{m=1}^{M} \boldsymbol{a}_m \boldsymbol{b}_m \leq \sqrt{\sum_{m=1}^{M} \boldsymbol{a}_m^2} \sqrt{\sum_{m=1}^{M} \boldsymbol{b}_m^2} \tag{24}$$

and letting $\boldsymbol{b}_m^2 = 1/M$, then the inequality in Eqn 24 becomes

$$\left(\frac{1}{\sqrt{M}} \sum_{m=1}^{M} \boldsymbol{a}_m \right)^2 \leq \sum_{m=1}^{M} \boldsymbol{a}_m^2. \tag{25}$$

Thus, the lower bound of Eqn 16 becomes

$$\|\boldsymbol{x}'_i - \boldsymbol{x}'_j\|_2^2 = \sum_{m=1}^{M} (\sum_{k=1}^{K} < \boldsymbol{\theta}_m, \boldsymbol{x}_i^k - \boldsymbol{x}_j^k >)^2 \tag{26}$$

$$\geq \frac{1}{M} \left(\sum_{m=1}^{M} \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \boldsymbol{x}_i^k - \boldsymbol{x}_j^k > \right)^2 \tag{27}$$

$$= \frac{1}{M} < \sum_{m=1}^{M} \boldsymbol{\theta}_m, \sum_{k=1}^{K} \boldsymbol{x}_i^k - \boldsymbol{x}_j^k >^2 . \tag{28}$$

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

$$\|\boldsymbol{x}'_i - \boldsymbol{x}'_j\|_2^2 \geq \frac{1}{M} < \boldsymbol{\phi}, \boldsymbol{z} >^2 \tag{29}$$

$$= \frac{1}{M} (\sum_{l=1}^{d} \boldsymbol{\phi}_l, \boldsymbol{z}_l)^2 \tag{30}$$

$$= \frac{1}{M} \sum_{l=1}^{d} \sum_{n=1}^{d} \boldsymbol{\phi}_l \boldsymbol{\phi}_n \boldsymbol{z}_l \boldsymbol{z}_n. \tag{31}$$

Then taking the expectation on both sides, the inequality becomes

$$\mathbb{E}[\|\boldsymbol{x}_i' - \boldsymbol{x}_j'\|_2^2] \geq \mathbb{E}[\frac{1}{M}\sum_{l=1}^{d}\sum_{n=1}^{d}\boldsymbol{\phi}_l\boldsymbol{\phi}_n\boldsymbol{z}_l\boldsymbol{z}_n] \tag{32}$$

$$= \frac{1}{M}\sum_{l=1}^{d}\sum_{n=1}^{d}\mathbb{E}[\boldsymbol{\phi}_l\boldsymbol{\phi}_n]\boldsymbol{z}_l\boldsymbol{z}_n. \tag{33}$$

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

$$\mathbb{E}[\boldsymbol{\phi}_l\boldsymbol{\phi}_n] = \begin{cases} 0 & l \neq n \\ M\sigma^2 & l = n \end{cases}. \tag{34}$$

Thus, the lower bound becomes

$$\mathbb{E}[\|\boldsymbol{x}_i' - \boldsymbol{x}_j'\|_2^2] \geq \frac{1}{M}\sum_{l=1}^{d}M\sigma^2\boldsymbol{z}_l^2 \tag{35}$$

$$= \sigma^2\|\boldsymbol{z}\|_2^2 \tag{36}$$

$$= \sigma^2\|\sum_{k=1}^{K}\boldsymbol{x}_i^k - \boldsymbol{x}_j^k\|_2^2 \tag{37}$$

$$= \sigma^2K^2\|\frac{1}{K}\sum_{k=1}^{K}\boldsymbol{x}_i^k - \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{x}_j^k\|_2^2. \tag{38}$$

Thus, the distance between two points after group convolution is lower bounded by the neighborhood centroid distance of the corresponding points before group convolution up to a scaling factor.

□

C ALGORITHMS

In this section, we provide the detailed algorithms of accelerating DGCNN, which is designed based on the proposed card shuffling theorems and hypotheses in Section 3. Algorithm 1 explains how the KNN procedure is simplified according to Theorem 1. Instead of conducting KNN search for each EdgeConv like DGCNN, only one full-fledged KNN search is conducted at the very beginning.

Algorithm 1: Accelerating the multiple KNN searches in the EdgeConvs.

Input: 3D point cloud $\mathcal{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$
Result: the K nearest neighbors used by all EdgeConvs.
Do one KNN search and keep L nearest neighbors, $L > K$.
for all i $\leq$ *number of EdgeConvs* **do**
  if i == 1 **then**
    Get the K nearest neighbors from the L neighbors.
  else
    Randomly sample K neighbors from the L neighbors.
  end if
end for

Algorithm 2 explains our shuffling strategy for feature gathering and convolution operation. As it is proved in Theorem 2, shuffling the order of neighbor feature gathering and the one by one convolution keeps the equivalent architecture for the basic EdgeConv. As a result, we can gather the edge features after applying two MLPs to the original points, and then add up the point and neighbor features.

Algorithm 2: Shuffling the order of feature gathering and convolution in EdgeConv.

1: **Input:** Input points $\mathcal{P} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$
2: **Result:** the K nearest neighbors used by all EdgeConvs.
3: Apply two MLPs $\boldsymbol{\theta}_m$ and $\boldsymbol{\psi}_m$ to the original points.
`// This leads to` $\boldsymbol{\theta}_m \cdot \boldsymbol{x}$ `and` $\boldsymbol{\psi}_m \cdot \boldsymbol{x}$`. Omitting the subscripts means that the same operation is done for every point.`
4: Do one neighbor search.
`// The neighbor search could be simplified as in Algorithm 1.`
5: Gather the edge features $\boldsymbol{\theta}_m \cdot \boldsymbol{x}_k$ for every point $\boldsymbol{x}_i$ according to the neighbor search.
6: Adds up the neighbor feature $\boldsymbol{\theta}_m \cdot \boldsymbol{x}_j$ and the point feature $\boldsymbol{\psi}_m \cdot \boldsymbol{x}_i$.

D Implementation Details

D.1 Classification of Point Cloud

For classification task, we used ModelNet40 dataset. ModelNet40 dataset consists of 12,311 meshed CAD models of 40 categories. We follow the experimental setting of PointNet (Qi et al., 2017a;b) and DGCNN (Wang et al., 2019). In order to evaluate our performance, we uniformly sample points of different numbers from the mesh faces to formulate point clouds. For the sake of fair comparison, the results reported in Table 1 are averaged from five runs. The number of parameters of all the networks is 1.8k. Inference runtime is measured on a single Titan Xp GPU and the batch size is reduced to 16 for running with 2048 points.

D.2 Segmentation of Point Cloud

For part segmentation task, we used ShapeNetPart dataset. ShapeNetPart consists of 16 object categories of 16,881 3D shapes, annotated with 50 parts. 2048 points are sampled from each shape. For semantic segmentation task, we used Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS). S3DIS consists of indoor scenes of 272 rooms in six inddor areas, annotated with 13 semantic categories. Our experiments follow the standard training, validation, and test split of DGCNN. For the sake of fairness comparison, the results reported in Table 2 are averaged from five runs. Part segmentation experiments are run on two Titan Xp GPUs and the batch size is 32.

D.3 Surface Reconstruction of Point Cloud

The visualization of meshes is done on the platform Open3D (Zhou et al., 2018).

E Feature Visualization

In order to validate that the neighborhood geometric feature is preserved after all the operations and shuffles, experiments are designed by extracting and visualizing feature map as a distance colormap rendered on the 3D point cloud. By observing Figure 2, we notice that the yellow part has a trend of converging to a smaller region with the propagation of the network within one epoch. By observing Figure 3, we can come to the conclusion that such convergence trend grows as the iterations moves on. At epoch 250 when the loss of the classification neural network converged, the yellowish neighbor features also converge into a very small region, and this region is smaller than the KNN represented by the green points. Figure 5 shows more results of feature space.

F Additional Experimental Results

In this section, we show additional experimental results of the surface reconstruction from point clouds. Figure 6 shows the computational resource comparison for the task of semantic segmentation. Figure 7 presents the ablation study of performance for semantic segmentation *w.r.t.* a wide range of parameters. Figure 8 shows more qualitative results for surface reconstruction.

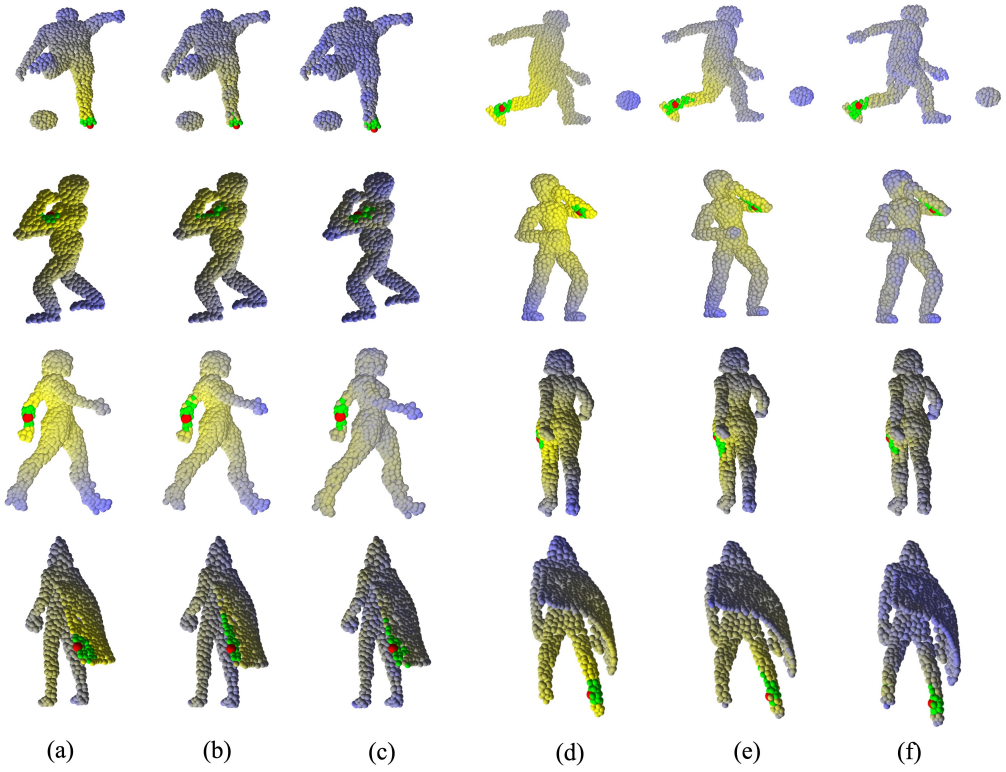
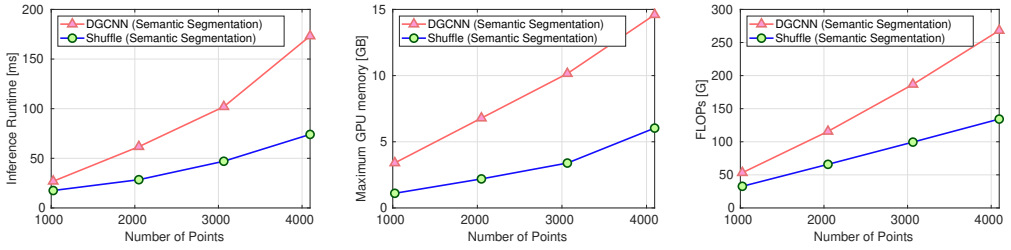
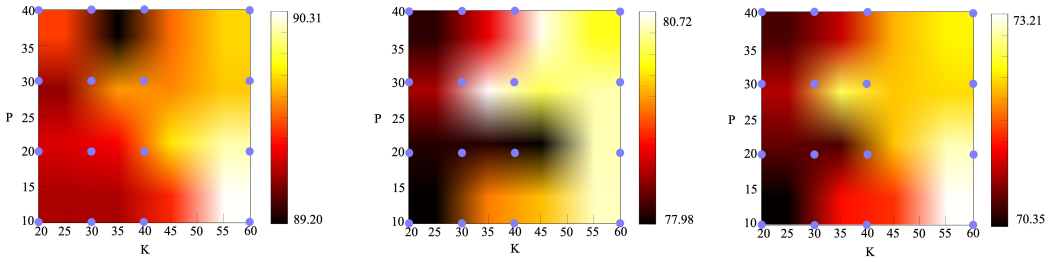


Figure 5: Renderings of input space and feature space as colormap between the red point and the rest of the points on ModelNet40 dataset. The green points represent KNN of the red point. (a) represents the input space. (b) represents the feature space extracted from the second layer of the network. (c) represents the feature space extracted from the last layer of the network. (d), (e),and (f) respectively follows the same layout with (a), (b), and (c).



(a) Runtime (b) GPU memory (c) FLOPs

Figure 6: Comparison between DGCNN and the proposed method (Shuffle) on point cloud semantic segmentation. (a) Runtime, (b) GPU memory consumption, and (c) FLOPs are reported for comparison.The proposed method can achieve significant reduction of computation resources.

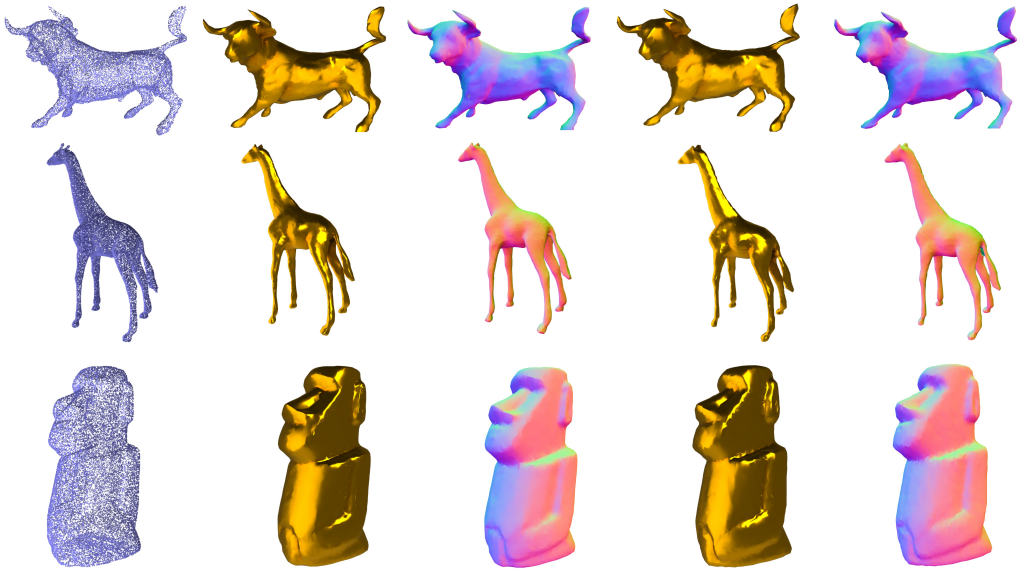


(a) Heatmap of overall acc. *w.r.t.* parameters K and P.

(b) Heatmap of balanced acc. *w.r.t.* parameters K and P.

(c) Heatmap of mIoU *w.r.t.* parameters K and P.

Figure 7: (a) Ablation study of overall acc. *w.r.t* parameters K and P. Values calculated are the points on the grid, and the hotmap is derived by bilinear interpolation. (b) and (c) follows the same layout with (e) for balanced accuracy and mean IoU.



(a) (b) (c) (d) (e)

Figure 8: Qualitative results of surface reconstruction. (a) Input point cloud. (b), (c) Surface and normal map reconstructed by Point2Mesh. (d), (e) Surface and normal map reconstructed by our method.