Revisiting 2D Convolutional Neural Networks for Graph-based Applications

Yecheng Lyu, *Student Member, IEEE,* Xinming Huang, *Senior Member, IEEE,* and Ziming Zhang, *Member, IEEE*

Abstract—Graph convolutional networks (GCNs) are widely used in graph-based applications such as graph classification and segmentation. However, current GCNs have limitations on implementation such as network architectures due to their irregular inputs. In contrast, convolutional neural networks (CNNs) are capable to extract rich features from large-scale input data, but they do not support general graph inputs. To bridge the gap between GCNs and CNNs, in this paper we study the problem of how to effectively and efficiently map general graphs to 2D grids that CNNs can be directly applied to, while preserving graph topology as much as possible. We therefore propose two novel graph-to-grid mapping schemes, namely, *graph-preserving grid layout (GPGL)* and its extension *Hierarchical GPGL (H-GPGL)* for computational efficiency. We formulate the GPGL problem as an integer programming and further propose an approximate yet efficient solver based on a penalized Kamada-Kawai method, a well-known optimization algorithm in 2D graph drawing. We propose a novel vertex separation penalty that encourages graph vertices to lay on the grid without any overlap. Along with this image representation, even extra 2D maxpooling layers contribute to the PointNet, a widely applied point-based neural network. We demonstrate the empirical success of GPGL on general graph classification with small graphs and H-GPGL on 3D point cloud segmentation with large graphs, based on 2D CNNs including VGG16, ResNet50 and multi-scale maxout (MSM) CNN.

Index Terms—graph neural network, convolutional neural network, graph classification, 3D point cloud segmentation

1 Introduction

Graph data processing using neural networks has been broadly attracting more and more research interests recently. Graph convolutional networks (GCNs) [?], [1], [2], [3], [4], [5], [6], [7], [8] [?], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26] [?], [27], [28], [29], [30], [31], [32] are a family of graph-based neural networks that extend convolutional neural networks (CNNs) to extract local features in general graphs with irregular input structures. The irregularity of a graph, including the orderless nodes and connections, however, makes the GCNs difficult to design as well as training from local patterns. In general, a GCN has two key operations to compute feature representations for the nodes in a graph, namely aggregation and transformation. That is, the feature representation of a node is computed as an aggregate of the feature representations of its neighbors before it is transformed by applying the weights and activation functions. To deal with graph irregularity the adjacency matrix is fed into the aggregation function to encode the topology. The weights are shared by all the nodes to perform convolutions to their neighborhood nodes.

CNNs, equipped with tens or hundreds of layers and millions of parameters thanks to their well designed operators upon 2D grid inputs, succeed in many research areas such as speech recognition [33] and computer vision [34]. In the grid inputs such as images, the highly ordered adjacency and unique layout bring the ability of designing accurate, efficient and salable convolutional kernels and pooling operations over large-scale data. How to apply CNNs to general graph inputs, however, still remains elusive.

- *Yecheng Lyu, Dr. Xinming Huang and Dr. Ziming Zhang are with the Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, Worcester, MA 01609, USA.*
  Email: {ylyu,xhuang,zzhang15}@wpi.edu
- *Part of this work was done when Lyu and Zhang worked at Mitsubishi Electric Research Laboratories (MERL).*

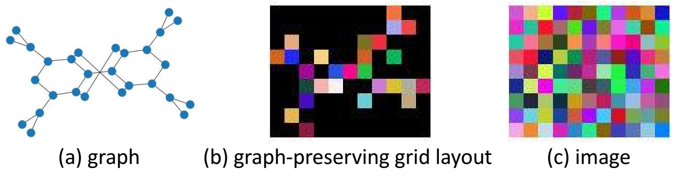


Figure 1: Illustration of differences between **(a)** a graph, **(b)** a graph-preserving grid layout (GPGL) of the graph, and **(c)** an image. The black color in (b) denotes no correspondence to any vertex in (a), and other colors denote non-zero features on the grid vertices.

Motivation. The key differences between graph convolution and 2D spatial convolution make CNNs much easier and richer in deep architectures than GCNs as well as being trained much more efficiently. Most of previous works in the literature such as [8], [12], [16] concentrate on handling the convolution weights based on the node connections, leading to high computation. Intuitively there also exists another way to process each graph by projecting the graph nodes onto a 2D grid so that CNNs can be employed directly. Such a method can easily benefit from CNNs in terms of network design and training in large scale. However, there are few existing works to explore this type of approaches. As graph topology can be richer and more flexible than grid for encoding important message of the graph, how to preserve graph topology on the grid becomes a key challenge in algorithm development.

Therefore, in order to bridge the gap between GCNs and CNNs, in contrast to previous works on generalizing the basic operations in CNNs to graph inputs, in this paper we mainly focus on studying the problem of **how to use CNNs as backbone for graph-based applications effectively and efficiently**. We then propose a principled method for projecting undirected graphs onto

0162-8828 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Worcester Polytechnic Institute. Downloaded on September 30,2021 at 20:54:18 UTC from IEEE Xplore. Restrictions apply.

the 2D grid with graph topology preservation.

In fact, the visualization of graphs in 2D space has been well studied in *graph drawing*, an area of mathematics and computer sciences whose goal is to present the nodes and edges of a graph on a plane with some specific properties (*e.g.* minimizing edge crossings [35], [36], minimizing the graph-node distance between graph domain and 2D domain [37], [38], showing possible clusters among the nodes [39]). In the literature, the *Kamada-Kawai (KK)* algorithm [37] is one of the most widely-used undirected graph visualization techniques. In general, the KK algorithm defines an objective function that measures the energy of each graph layout *w.r.t.* some theoretical graph distance, and searches for the (local) minimum that gives a reasonably good 2D visualization of the graph regardless the distances among the nodes.

As described above, we can see that graph drawing algorithms may play an important role in connecting graph applications with CNNs for geometric deep learning (GDL), in general. To the best of our knowledge, however, such graph drawing algorithms have never been explored for GDL. One possible reason is that graph drawing algorithms often work in continuous spaces, while our case requires *discrete* spaces (*i.e.* grid) where CNNs can be deployed. Overall, how to project graphs onto the grid with topology preservation for GDL is still elusive in the literature.

Contributions. To address the problem above, in this paper we propose a novel *graph-preserving grid layout* (GPGL), an integer programming problem that minimizes the topological loss on the 2D grid so that CNNs can be used for GDL on undirected graphs. Technically solving such a problem is very challenging because potentially one needs to solve a highly nonconvex optimization problem in a discrete space. We manage to do so effectively by proposing a penalized KK method with a novel vertex separation penalty, followed by the rounding technique. As a result, our GPGL algorithm can approximately preserve the irregular structural information in a graph on the regular grid as graph layout, as illustrated in Fig. 1. To further improve the computational efficiency of GPGL, we also propose a hierarchical GPGL (H-GPGL) algorithm as an extension to handle large graphs.

In summary, our key contributions of this paper are as follows:

- We are the *first*, to the best of our knowledge, to explicitly explore the usage of graph drawing algorithms in the context of GDL, and accordingly propose a novel GPGL algorithm and its variance H-GPGL to project graphs onto the 2D grid with minimum loss in topological information.
- We demonstrate the empirical success of GPGL on graph classification with small graphs, and H-GPGL on 3D point cloud segmentation with large graph. In the experiment, we clearly shows that PointNet with 2D max-pooling layers and 2D CNNs including VGG16, ResNet50 and multi-scale maxout(MSM) CNN benefit from the GPGL image representation and gain a large improvement over PointNet, a point-wised neural network.

2 Related Work

Graph Drawing & Network Embedding. Graph drawing can be considered as a subdiscipline of network embedding [40], [41], [42] whose goal is to find a low dimensional representation of the network nodes in some metric space so that the given similarity (or distance) function is preserved as much as possible. In summary, graph drawing focuses on the 2D/3D visualization of graphs [43], [44], [45], [46], [47], while network embedding emphasizes the learning of low dimensional graph representations. Despite the research goal, similar methodology has been applied to both areas. For instance, the KK algorithm [37] was proposed for graph visualization as a force-based layout system with advantages such as good-quality results and strong theoretical foundations, but suffering from high computational cost and poor local minima. Similarly [48] proposed a global geometric framework for network embedding to preserve the intrinsic geometry of the data as captured in the geodesic manifold distances between all pairs of data points. There are also some works on drawing graphs on lattice, *e.g.* [49].

In contrast to graph drawing, our focus is to project an existing graph onto the grid with minimum topological loss so that CNNs can be deployed efficiently and effectively to handle graph data. In such a context, we are not aware of any work in the literature that utilizes the graph drawing algorithms to facilitate GDL, to the best of our knowledge.

Graph Synthesis & Generation. Methods in this field, *e.g.* [50], [51], [52], [53], often aim to learn a (sophisticated) generative model that reflects the properties of the training graphs. Recently, [54] proposed learning an encoder-decoder for the graph layout generation problem to systematically visualize a graph in diverse layouts using deep generative model. [55] proposed jointly learning the graph structure and the parameters of GCNs by approximately solving a bilevel program that learns a discrete probability distribution on the edges of the graph for classification problems.

In contrast to such methods above, our algorithm for GPGL is essentially a self-supervised learning algorithm that is performed for each individual graph and requires no training at all. Moreover, we focus on re-deploying each graph onto the grid as layout while preserving its topology. This procedure is separate from the training of CNNs later.

Geometric Deep Learning. In general GDL studies the extension of deep learning techniques to graph and manifold structured data (*e.g.* [2], [4], [56], [57]). In particular in this paper we focus on graph data only. Broadly GDL methods can be categorized into spatial methods (*e.g.* [56], [58], [59]) and spectral methods (*e.g.* [1], [60], [61]). Some nice survey on this topic can be found in [**?**], [**?**], [4], [40].

[12] proposed a framework for learning convolutional neural networks by applying the convolution operations to the locally connected regions from graphs. We are different from such a work by applying CNNs to the grids where graphs are projected to with topology preservation. [**?**] proposed an ad-hoc method to project graphs onto 2D grid and utilized CNNs for graph classification. Specifically each node is embedded into a high dimensional space, then mapped to 2D space by PCA, and finally quantized into grid. In contrast, we propose a principled and systematical way based on graph drawing, *i.e.* a nonconvex integer programming formulation, for mapping graphs onto 2D grid. Besides, our graph classification performance is much better than both works. On MUTAG and IMDB-B data sets we can achieve 94.18% and 74.9% test accuracy with 5.23% improvement over [12] and 4.5% improvement over [**?**], respectively.

3D Point Cloud Processing. Point clouds can be treated as graphs with undetermined connections. In point cloud processing, several existing works such as [62] built the adjacency matrix using the K-nearest-neighbor search and then applied GCNs to the generated graph. Some other works like [63] used adaptive search ranges

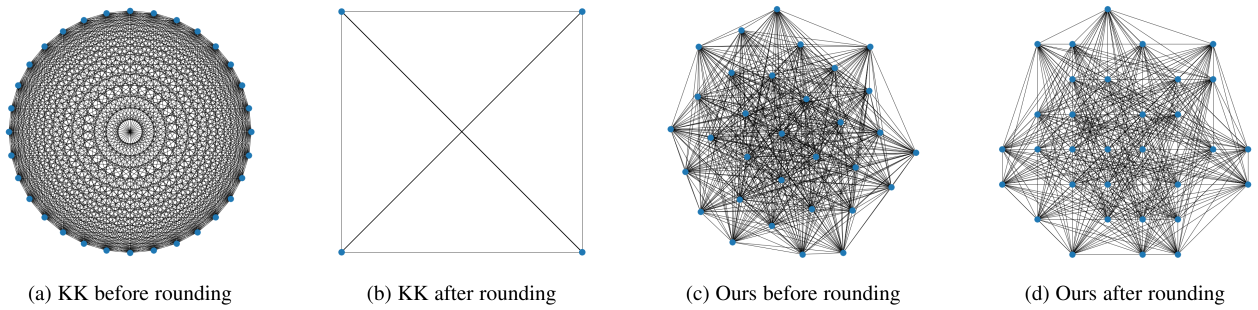

(a) KK before rounding (b) KK after rounding (c) Ours before rounding (d) Ours after rounding

Figure 2: Illustration of layout comparison between the KK algorithm and our proposed penalized KK algorithm before and after rounding based on a fully-connected graph with 32 vertices.

to generate the adjacency matrix and then applied GCNs to it. Point cloud processing, same as graph processing, suffers from the limited size and efficiency of GCN operators.

3D point cloud segmentation, as one of the key applications using point clouds, has attracted increasing research attention in the recent past. The existing works can be summarized into two major groups: point-based approaches and graph-based approaches. In point-based approaches, the pioneering work PointNet [64] applied fully connected layers to each single point and then extracted the global features by aggregating all the point-wise features. By its design, PointNet effectively gathers the global features from all points and broadcasts them back to all point for point-wise semantic prediction. However, the accuracy is limited due to the lack of local feature extraction. Later on, some other solutions are proposed to introduce the local feature extractions. PointNet++ [65], the succeeding work of PointNet, hierarchically grouped the points in local regions and extracted the local features along the hierarchy. PointCNN [66], SpiderCNN [67], So-Net [68], RS-CNN [69], PointConv [70], ψ-CNN [71], A-CNN [72], ShellNet [73], DensePoint [74],SPLATNet3D [75] proposed their local point grouping and feature extraction operators, resulting in high accuracy.

The graph-based approaches, on the other hand, construct a connected graph from each point cloud and then apply GCNs to it. Kd-Net [76], RGCNN [63], FeaStNet [77], KC-Net [78], DGCNN [79], LDGCNN [62] are some good works in this branch. However, the complexity of graph neural convolution kernel limits their capability of achieving outstanding accuracy.

Besides, there are several works trying to project the point clouds into 2D grid maps and apply 2D convolution to it. SFCNN [80] projected each point cloud into a spherical surface and applied a fractal convolution operation to it. InterpConv [81] creatively interpolated the graph convolution into a 2D convolution upon the adjacent grid cells. Lyu *et al.* [82] solves the point cloud segmentation problem by transferring the point cloud into image space and apply U-Net on it. Those three approaches make good effort to introduce the 2D convolution into point cloud processing. However, they are yield to customized 2D convolution kernels that does not support commonly used feature extraction backbones such as VGG [83], ResNet [34] and Xception [84].

In our approach, we follow the graph-based approaches to construct a connected graph upon each point cloud. In contrast, we further project the graph nodes onto a 2D grid map using H-GPGL. In this way each point in the point cloud is assigned to one of the nodes in the 2D grid and we can now apply CNNs to the 2D graph layouts as we do on images.

3 Graph-Preserving Grid Layout (GPGL)

3.1 Problem Setup

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with a vertex set $\mathcal{V}$ and an edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, and $s_{ij} \geq 1, \forall i \neq j$ be the graph-theoretic distance such as shortest-path between two vertices $v_i, v_j \in \mathcal{V}$ on the graph that encodes the graph topology.

Now we would like to learn a function $f : \mathcal{V} \rightarrow \mathbb{Z}^2$ to map the graph vertex set to a set of 2D integer coordinates on the grid so that the graph topology can be preserved as much as possible given a metric $d : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$ and a loss $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$. As a result, we are seeking for f to minimize the following objective:

$$\min_f \sum_{i \neq j} \ell(d(f(v_i), f(v_j)), s_{ij}). \tag{1}$$

Now letting $\mathbf{x}_i = f(v_i) \in \mathbb{Z}^2$ as reparametrization, we can rewrite Eq. 1 as the following *integer programming* problem:

$$\min_{\mathcal{X} \subseteq \mathbb{Z}^2} \sum_{i \neq j} \ell(d(\mathbf{x}_i, \mathbf{x}_j), s_{ij}), \tag{2}$$

where the set $\mathcal{X} = \{\mathbf{x}_i\}$ denotes the *2D grid layout* of the graph, *i.e.* all the vertex coordinates on the 2D grid.

Self-Supervision. Note that the problem in Eq. 2 needs to be solved for each individual graph, which is related to self-supervision as a form of unsupervised learning where the data itself provides the supervision [85]. This property is beneficial for data augmentation, as every local minimum will lead to a grid layout for the same graph.

2D Grid Layout. In this paper we are interested in learning only 2D grid layouts for graphs, rather than higher dimensional grids (even 3D) where we expect that the layouts would be more compact in volume and would have larger variance in configuration, both bringing more challenges into training CNNs properly later. We confirm our hypothesis based on empirical observations. Besides, the implementation of 3D basic operations in CNNs such as convolution and pooling are often slower than 2D counterparts, and the operations beyond 3D are not available publicly.

Relaxation & Rounding for Integer Programming. Integer programming is NP-complete and thus finding exact solutions is challenging, in general [86]. Relaxation and rounding is a widely used heuristic for solving integer programming due to its efficiency [87], where the rounding operator is applied to the solution from the real-number relaxed problem as the solution for the integer programming. In this paper we employ this heuristic to learn 2D grid layouts. For simplicity, in the sequel we will only discuss how to solve the relaxation problem (*i.e.* before rounding).

3.2 Penalized Kamada-Kawai Algorithm for GPGL

In this paper we set ℓ and d in Eq. 2 to the least-square loss and Euclidean distance to preserve topology, respectively, so that we can develop new algorithms based on the classic KK algorithm.

3.2.1 Preliminary: Kamada-Kawai Algorithm

The KK graph drawing algorithm [37] was designed for a (relaxed) problem in Eq. 2 with a specific objective function as follows:

$$\min_{\mathcal{X}\subseteq\mathbb{R}^2} \mathcal{L}_{KK} = \sum_{i\neq j} \frac{1}{2}\left(\frac{d_{ij}}{s_{ij}} - 1\right)^2, \tag{3}$$

where $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|, \forall (i,j)$ denotes the Euclidean distance between vertices v_i and v_j. Note that there is no regularization/penalty to control the distribution of nodes in 2D visualization.

Fig. 2 illustrates the problems using the KK algorithm when projecting the fully-connected graph onto 2D grid. Eventually KK learns a circular distribution with equal space among the vertices as in Fig. 2(a) to minimize the topology preserving loss in Eq. 3. When taking a close look at these 2D locations we find that after transformation all these locations are within the square area $[0,1]\times[0,1]$, leading to the square pattern in Fig. 2(b) after rounding. Such behavior totally makes sense to KK because it does not care about the grid layout but only the topology preserving loss. However, our goal is not only to preserve the topology but also to make graphs visible on the 2D grid in terms of vertices.

3.2.2 Our Algorithm

To this end, we propose a penalized KK algorithm as listed in Alg. 1, which tries to minimize the following penalized KK loss:

$$\min_{\mathcal{X}\subseteq\mathbb{Z}^2} \mathcal{L}_{GPGL} = \mathcal{L}_{KK} + \mathcal{L}_{sep}. \tag{4}$$

Vertex Separation Penalty. We propose a novel vertex separation penalty to regularize the vertex distribution on the grid. The intuition behind it is that when the minimum distance among all the vertex pairs is larger than a threshold, say 1, it will guarantee that after rounding every vertex will be mapped to a unique 2D location with no overlap. But when any distance is smaller than the threshold, it should be considered to enlarge the distance, otherwise, no penalty. Moreover, we expect that the penalties will grow faster than the change of distances and in such a way the vertices can be re-distributed more rapidly. Based on these considerations we propose the following penalty:

$$\mathcal{L}_{sep} = \lambda \sum_{i\neq j} \max\left\{0, \frac{\alpha}{d_{ij}} - 1\right\}, \tag{5}$$

where $\alpha \geq 0, \lambda \geq 0$ are two predefined constants. From the gradient of $\mathcal{L}_{sep}$ *w.r.t.* an arbitrary 2D variable $\mathbf{x}_i$, that is,

$$\frac{\partial \mathcal{L}_{sep}}{\partial \mathbf{x}_i} = -\lambda \sum_{i\neq j} \frac{\mathbf{x}_i - \mathbf{x}_j}{d_{ij}^3} \mathbf{1}_{\{d_{ij}<\alpha\}} \tag{6}$$

where $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function returning 1 if the condition is true, otherwise 0, we can clearly see that α as a threshold controls when penalties occur, and λ controls the trade-off between the two losses, leading to different step sizes in gradient based optimization.

Initialization. Note that both KK and our algorithms are highly nonconvex, and thus good initialization is need to make both work well, *i.e.* convergence to good local minima.

Algorithm 1 Penalized Kamada-Kawai Algorithm for GPGL

```
Input  : undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parameters $\alpha, \lambda$
Output: 2D grid layout $\mathcal{X}^*$
Compute graph distance $\{s_{ij}\}$;
$\tilde{\mathcal{X}} \leftarrow \arg\min_{\mathcal{X}} \mathcal{L}_{KK}$ with a (randomly shuffled) circular layout;
$\mathcal{X}^* \leftarrow \arg\min_{\mathcal{X}\subseteq\mathbb{R}^2} \mathcal{L}_{GPGL}$ with set $\tilde{\mathcal{X}}$ as initialization;
$\mathcal{X}^* \leftarrow \text{round}(\mathcal{X}^*)$;
return $\mathcal{X}^*$;
```

To this end, we first utilize the KK algorithm to generate a vertex distribution. To do so, we employ the implementation in the Python library NetworkX [88] which uses a circular layout as initialization by default. By default setting, L-BFGS-B Nonlinear Optimization is implemented to optimize Eqn. 4. As discussed above, KK has no control on the vertex distribution. This may lead to serious **vertex loss** problems in the 2D grid layout where some of vertices in the original graph merge together as a single vertex on the grid after rounding due to small distances (see our experiments).

Topology Preservation with Penalty. As we observe, the key challenge in topology preservation comes from the node degree, and the lower degree the easier for preservation. Since there are only 8 neighbors at most in the 2D grid layout, it will induce a penalty for a graph vertex whose degree is higher than 8. Fig. 2 illustrates such a case where the original graph is full-connected with 32 vertices. With the help of our proposed penalty, we manage to map this graph to a ball-like grid layout, as shown in Fig. 2(c) and (d). Besides we have the following proposition to support such observations:

Proposition 1. *An ideal 2D grid layout with no vertex loss for a full-connected graph with $|\mathcal{V}|$ vertices is a ball-like shape with radius of $\lceil(\frac{|\mathcal{V}|}{\pi})^{\frac{1}{2}}\rceil$ that minimizes Eq. 2 with relaxation of the penalized Kamada-Kawai loss. Here $\lceil\cdot\rceil$ denotes the ceiling operation.*

Proof. Given the conditions in the proposition above, we have $s_{ij} = 1, d_{ij} \geq 1, \forall i \neq j$ and $\mathcal{L}_{sep} = 0$. Without loss of generalization, we uniformly deploy the graph vertices in a circle and set the circular center A to a node on the 2D grid. Now imagine the gradient field over all the vertices as a sandglass centered at A where each vertex is a ball with a unit diameter. Then it is easy to see that by the "gravity" (*i.e.* gradient) all the vertices move towards the center A, and eventually are stabilized (as a local minimum) within an r-radius circle whose covering area should satisfy $|\mathcal{V}| \leq \pi r^2$, *i.e.* $r = \lceil(\frac{|\mathcal{V}|}{\pi})^{\frac{1}{2}}\rceil$ as the smallest sufficient radius to cover all the vertices with guarantee. We now complete our proof. □

Note that Fig. 2(d) exactly verifies Prop. 1 with a radius $r = \lceil(\frac{32}{\pi})^{\frac{1}{2}}\rceil = 4$. In summary, our algorithm can (approximately) manage to preserve graph topology on the 2D grid even when the node degree is higher than 8.

Computational Complexity. The KK algorithm has the complexity of, at least, $O(|\mathcal{V}|^2)$ [38] that limits the usage of KK to medium-size graphs (*e.g.* 50-500 vertices). Since our algorithm in Alg. 1 is based on KK, it unfortunately inherits this limitation as well. To accelerate the computation for large-scale graphs, we potentially can adopt the strategy in multi-scale graph drawing algorithms such as [89]. However, such an extension is out of scope of this paper, and we will consider it in our future work.

3.3 Experiments: Small Graph Classification

3.3.1 Data Sets & Data Augmentation

We evaluate our method, *i.e.* GPGL + (multi-scale maxout) CNNs, on four medium-size benchmark data sets for graph classification, namely MUTAG, IMDB-B, IMDB-M and PROTEINS. Table 1 summarizes some statistics of each data set. Note that the max node degree on each data set is at least 8, indicating that ball-like patterns as discussed in Prop. 1 may occur, especially for IMDB-B and IMDB-M.

As mentioned in self-supervision, each local minimum from our penalized KK algorithm in Alg. 1 will lead to a grid layout for the graph, while each minimum depends on its initialization. Therefore, to augment grid layout data from graphs, we do a random shuffle on the circular layout when applying Alg. 1 to an individual graph. Fig. 3 illustrates two augmented image representations from a graph sample in MUTAG dataset.

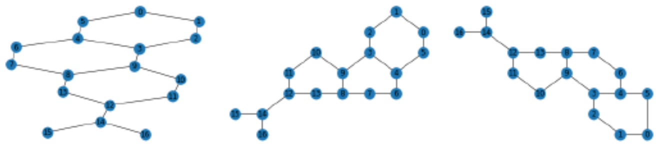

(a) Input graph (b) Output image 1 (c) Output image 2

Figure 3: Illustration of augmented image representations from a graph using GPGL

3.3.2 Grid-Layout based 3D Representation

Once a grid layout is generated, we first crop the layout with a sufficiently large fixed-size window (*e.g.* 32×32), and then associate each vertex feature vector from the graph with the projected node within the window. All the layouts are aligned to the top-left corner of the window. The rest of nodes in the window with no association of feature vectors are assigned to zero vectors.

Once vertex loss occurs, we take an average, by default, of all the vertex feature vectors (*i.e.* average-pooling) and assign it to the grid node. We also compare average-pooling with max-pooling for merging vertices, and observe similar performance empirically in terms of classification.

3.3.3 Classifier: Multi-Scale Maxout CNNs (MSM-CNNs)

We apply CNNs to the 3D representations of graphs for classification. As we discussed above, once the node degree is higher than 8, the grid layout cannot fully preserve the topology, but rather tends to form a ball-like compact pattern with larger neighborhood. To capture such neighborhood information effectively, the kernel sizes in the 2D convolution need to vary. Therefore, the problem now boils down to a feature selection problem with convolutional kernels.

Considering these, here we propose using a multi-scale maxout CNN as illustrated in Fig. 6. We use consecutive convolutions with smaller kernels to approximate the convolutions with larger kernels. For instance, we use three 3×3 kernels to approximate a 7×7 kernel. The maxout [90] operation selects which scale per grid node is good for classification and outputs the corresponding features. Together with other CNN operations such as max-pooling, we can design deep networks, if necessary.

Table 1: Statistics of benchmark data sets for graph classification.

Data Set	Num. of Graph	Num. of Class	Avg. Node	Avg. Edge	Avg. Degree	Max Degree	Feat. Dim.
MUTAG	188	2	17.93	19.79	1.10	8	7
IMDB-B	1000	2	19.77	96.53	4.88	270	136
IMDB-M	1500	3	13.00	65.94	5.07	176	89
PROTEINS	1113	2	39.06	72.82	1.86	50	3

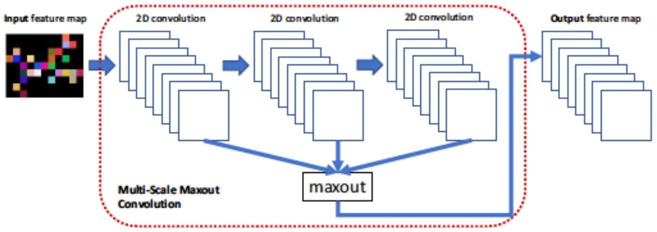


Figure 6: Multi-scale maxout convolution (MSM-Conv).

3.3.4 Implementation

By default, we set the parameters in Alg. 1 as $\alpha = 1.25, \lambda = 1000$. We crop all the grid layouts to a fixed-size 32×32 window. Also by default, for the MSM-CNNs we utilize three consecutive 3×3 kernels in the MSM-Conv, and design a simple network of "MSM-Conv(64)$\rightarrow$max-pooling$\rightarrow$MSM-Conv(128)$\rightarrow$max-pooling$\rightarrow$MSM-Conv(256)$\rightarrow$global-pooling$\rightarrow$FC(256)$\rightarrow$FC(128)" as hidden layers with ReLU activations, where FC denotes a fully connected layer and the numbers in the brackets denote the numbers of channels in each layer. We employ Adam [91] as our optimizer, and set batch size, learning rate, and dropout ratio to 10, 0.0001, and 0.3, respectively.

3.3.5 Ablation Study

Effects of α, λ on Grid Layout and Classification. To understand their effects on the 2D grid layout generation, we visualize some results in Fig. 4 using different combinations of α, λ. We can see that:

- From Fig. 4(a)-(c), the diameters of grid layouts are 5×5, 6×6, 7×7 for $\alpha = 1.00, 1.25, 1.50$, respectively. This strongly indicates that a smaller α tends to lead to a more compact layout at the risk of losing vertices.
- From Fig. 4(c)-(e), similarly the diameters of grid layouts are 5×5, 6×6, 6×6 for $\lambda = 200, 1000, 5000$, respectively. This indicates that a smaller λ tends to lead to a more compact layout at the risk of losing vertices as well. In fact in Fig. 4(d) node 1 and node 5 are merged together. When λ is sufficiently large, the layout tends to be stable.

Such observations follow our intuition in designing Alg. 1, and occur across all the four benchmark data sets.

We also test the effects on classification performance. For instance, we generate 21x grid layouts using data augmentation on MUTAG, and list our results in Table 2. Clearly our default setting achieves the best test accuracy. We also observe that in case $\lambda = 0$ where $\mathcal{L}_{sep} = 0$ and the KK algorithm works without vertex separation penalty, the classification accuracy is much worse than those with vertex separation penalty.

Vertex Loss, Graph Topology & Misclassification. To better understand the problem of vertex loss, we visualize some cases in Fig. 5. The reason for this behavior is due to the small distances among the vertices returned by Alg. 1 that cannot survive from

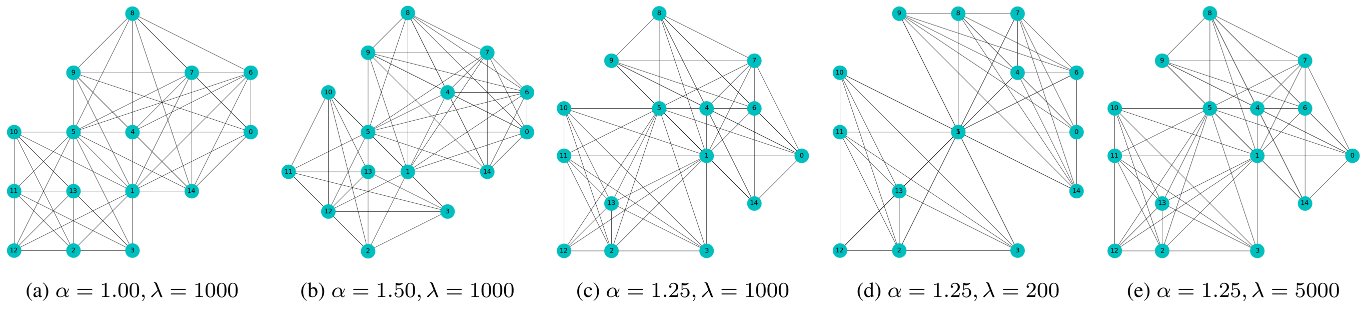

(a) $\alpha = 1.00, \lambda = 1000$ (b) $\alpha = 1.50, \lambda = 1000$ (c) $\alpha = 1.25, \lambda = 1000$ (d) $\alpha = 1.25, \lambda = 200$ (e) $\alpha = 1.25, \lambda = 5000$

Figure 4: Illustration of effects of different combinations of α, λ on grid layout generation (IMDB-B)

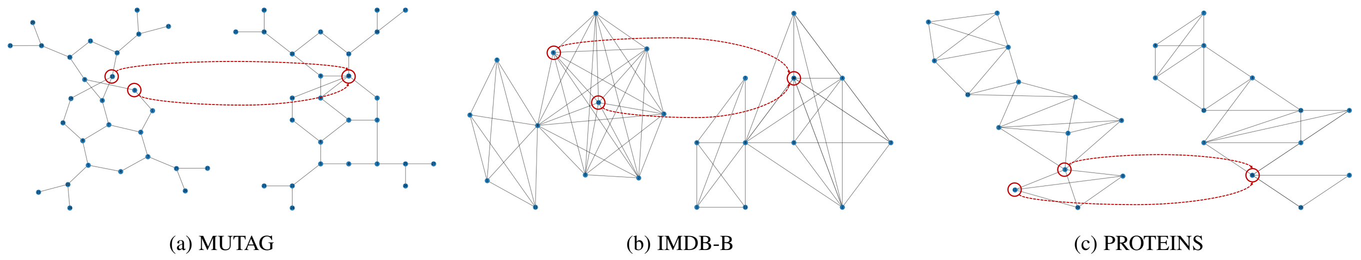

(a) MUTAG (b) IMDB-B (c) PROTEINS

Figure 5: Illustration of vertex loss on different data sets: In each subfigure, **(left)** before rounding and **(right)** after rounding.

Table 2: Mean accuracy (%) using different combinations of α, λ.

Data Set	$\alpha = 1.00$ $\lambda = 0$	$\alpha = 1.00$ $\lambda = 1000$	$\alpha = 1.50$ $\lambda = 1000$	$\alpha = 1.25$ $\lambda = 1000$	$\alpha = 1.25$ $\lambda = 200$	$\alpha = 1.25$ $\lambda = 5000$
MUTAG (21×)	80.51	85.14	83.04	86.31	85.26	85.26

Table 3: Vertex loss ratio (%) on each data set.

Data Set	MUTAG	IMDB-B	IMDB-M	PROTEINS
Vertex Loss	1.06	0.99	0.40	0.90

rounding. Unfortunately we do not observe a pattern on when such loss will happen. Note that our Alg. 1 cannot avoid vertex loss with guarantee, and in fact the vertex loss ratio on each data set is very low, as shown in Table 3.

Further we test the relationship between vertex loss and misclassification, and list our results in Table 4 where $\mathcal{G}_{v.l.}$, $\mathcal{G}_{n.v.l.}$, and $\mathcal{G}_{mis.}$ denote the sets of graphs with vertex loss, no vertex loss, and misclassification, respectively, $\cap$ denotes the intersection of two sets, $|\cdot|$ denotes the cardinality of the set, and the numbers in the brackets denote the numbers of grid layouts per graph in data augmentation. From this table, we can deduce that vertex loss cannot be the key reason for misclassification, because it takes only tiny portion in misclassification and the ratios of misclassified graphs with/without vertex loss are very similar, indicating that misclassification more depends on the classifier rather than vertex loss.

As discussed before, a larger node degree is more difficult for preserving topology. In this test we would like to verify whether such topology loss introduces misclassification. Compared with the statistics in Table 1, it seems that topology loss does cause trouble in classification. One of the reasons may be that the variance of the grid layout for a vertex with larger node degree will be higher due to perturbation. Designing better CNNs will be one of our future works to improve the performance.

CNN based Classifier Comparison. We test the effectiveness of our GPGL algorithm on MUTAG dataset with PointNet [66],

Table 4: Ratios (%) between vertex loss and misclassification.

Data Set	$\frac{\lvert\mathcal{G}_{v.l.}\rvert}{\lvert\mathcal{G}_{mis.}\rvert}$	$\frac{\lvert\mathcal{G}_{v.l.} \cap \mathcal{G}_{mis.}\rvert}{\lvert\mathcal{G}_{v.l.}\rvert}$	$\frac{\lvert\mathcal{G}_{n.v.l.} \cap \mathcal{G}_{mis.}\rvert}{\lvert\mathcal{G}_{n.v.l.}\rvert}$
MUTAG (21x)	1.06	20.00	16.70
IMDB-B (3x)	0.99	16.18	37.41
PROTEINS (3x)	0.90	24.32	29.89

PointNet with 2D max-pooling, VGG16 [83], ResNet50 [34], and our MSM-CNN. In those networks, PointNet is a point-based network that has no 2D convolution or pooling expect for a global max-pooling at last; by applying to an image representation, we add a 2D max-pooling after each point-wise convolution layer to introduce 2D integration to image pixels. VGG16, ResNet50 are widely used 2D CNNs for image classification. Our MSM-CNN is also a 2D CNN for image classification. In all test we use the same augmented data. From Table 5 we observe that 2D CNNs are significantly better than the PointNet with large margins of 21.66%, and PointNet improves 19.68% in accuracy by simply adding 2D max-pooling layers. The observation clearly shows that 2D CNNs benefit from our GPGL algorithm.

Table 5: Mean accuracy (%) using different CNN classifiers.

Network	PointNet	PointNet with 2D pooling	VGG16	ResNet50	MSM
MUTAG (101×)	66.55	86.23	88.21	91.02	94.18

Effect of Data Augmentation using Grid Layouts on Classification. In order to train the deep classifiers well, the amount of training data is crucial. As shown in Alg. 1, our method can easily generate tons of grid layouts that effectively capture different characteristics in the graph. Given the memory limit, we demonstrate the test performance for data augmentation in Fig. 8, ranging from 1x to 101x with step 10x. As we see clearly, data augmentation can significantly boost the classification accuracy on MUTAG, and similar observations have been made for the other data sets.

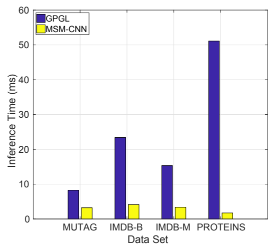


Figure 7: Running time at inference for GPGL and MSM-CNN.

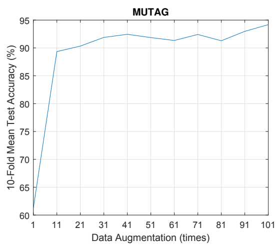


Figure 8: Illustration of data augmentation on classification.

Algorithm 2 H-GPGL Algorithm

```
Input : number of nodes for GPGL $N$, undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, parent spatial location $\mathbf{a}_p$, parent grid size $\mathbf{s}_p$, child grid size $\mathbf{s}_c$, 2D grid layout $\mathcal{X}^*$
Output: $\mathcal{X}^*$
$T \leftarrow round(\log_N |\mathcal{V}|)$;
if $T = 1$ then
    $(\mathcal{X}, \mathcal{A}_c) \leftarrow$ GPGL$(\mathcal{G}, \mathbf{a}_p, \mathbf{s}_c)$;
    $\mathcal{X}^* \leftarrow FitIntoGrid(\mathcal{X}^*, \mathcal{X}, \mathcal{A}_c)$;
else
    $\mathcal{H} \leftarrow partition(\mathcal{G}, N)$;
    Construct a connectivity graph $\mathcal{G}_H$ based on $\mathcal{H}$;
    $(\mathcal{X}_H, \mathcal{A}_H) \leftarrow$ GPGL$(\mathcal{G}_H, \mathbf{a}_p, \mathbf{s}_p)$;
    for $i = 1, \cdots, N$ do
        $\mathcal{X}^* \leftarrow$ H-GPGL$(N_i, \mathcal{H}_i, \mathcal{A}_{H_i}, \mathbf{s}_p, \mathbf{s}_c, \mathcal{X}^*)$;
    end
end
return $\mathcal{X}^*$;
```

3.3.6 State-of-the-Art Comparison

To do a fair comparison for graph classification, we follow the standard routine, *i.e.* 10-fold cross-validation with random split. All the comparisons are summarized in Fig. 9. We generate 101x, 21x, 11x, 11x of the original size of MUTAG, IMDB-B, IMDB-M, and PROTEINS for data augmentation, and achieves $94.18\% \pm 4.61\%, 74.90\% \pm 4.01\%, 68.67\% \pm 1.22\%, 79.52\% \pm 1.72\%$ in terms of test accuracy, respectively. Among the tests, we achieves the state-of-the-art accuracy in IMDB-M and PROTEINS dataset, and near the state-of-the-art accuracy in MUTAG and IMDB-B dataset.

4 Hierarchical GPGL (H-GPGL)

As we discussed before, the computational complexity of the KK algorithm is at least $O(|\mathcal{V}|^2)$ where $|\mathcal{V}|$ denotes the number of nodes in a graph. Also empirically from Fig. 7 we verify that even for small graphs the computational bottleneck of our method from GPGL (even this procedure can be offline) is significant that prevents ours from large-graph applications. For instance, the running time of GPGL on a simple graph (low node degree) with 2048 nodes takes about 90s. Therefore, in order to apply our method to large graphs we need to overcome this computational challenge. This motivates our hierarchical GPGL algorithm.

4.1 Algorithm Overview

Our basic idea is to partition each graph into a set of subgraphs that can be organized in a hierarchy such as (complete) m-way tree where each node presents a subgraph and each level preserves the original graph connectivity among the subgraphs at the level. In this way we can preserve the original graph information as much as possible. To do so, an intuitive way is to partition a graph recursively, as listed in Alg. 2. We also illustrate an example of two-level GPGL generation procedure in Fig. 10. Here we first partition the graph with 2048 nodes into 32 subgraphs and map the connectivity graph of these subgraphs onto a 16×16 grid using the GPGL algorithm in Alg. 1. Then we project each subgraph again onto a 16×16 grid, leading to a 256×256 grid layout shown on the right side.

Notations. We define $N \in \mathbb{R}$ (*resp.* $N_i, \forall i$) as the number of nodes that GPGL can easily deal with. In the hierarchy, $\mathbf{a}_p \in \mathbb{R}^2$ denotes the 2D spatial location of a parent node in the grid $\mathcal{X}^* \subseteq \mathbb{R}^2$ and $\mathcal{A}_c \subseteq \mathbb{R}^2$ denotes the set of 2D spatial locations of the child nodes of the parent node. $\mathcal{H}$ denotes the set of partitioned subgraphs satisfying $\bigcup \mathcal{H} = \mathcal{G}$, and $\mathcal{X}_H \subseteq \mathbb{R}^2, \mathcal{A}_H \subseteq \mathbb{R}^2$ denote the grid layout and the spatial location set for subgraph connectivity graph $\mathcal{G}_H$ of $\mathcal{H}$. $\mathcal{H}_i \subseteq \mathbb{R}^2, \mathcal{A}_{H_i} \subseteq \mathbb{R}^2, \forall i$ denote the i-th subgraph in $\mathcal{H}$ and its spatial location, respectively. Note that there is one more output of $GPGL$ in Alg. 2 than Alg. 1, which is the 2D location set for the input nodes. This operation can be easily implemented by adding an extra traversal on the grid.

Fitting a Graph Node into the Grid Layout. Let $\mathcal{P}_v$ denote the path from a leaf v (*i.e.* a graph node) to the root in the tree hierarchy through M internal nodes. Then the 2D location of v, $\mathbf{a}_v$, on the grid layout $\mathcal{X}^*$ can be easily computed as follows:

$$\mathbf{a}_v = \tilde{\mathbf{a}}_{\mathcal{P}_v^{(0)}} + \mathbf{s}_c \sum_{m=1}^{M} \tilde{\mathbf{a}}_{\mathcal{P}_v^{(m)}} \times (\mathbf{s}_p)^m, \qquad (7)$$

where $\tilde{\mathbf{a}}_{\mathcal{P}_v^{(m)}}, \forall m$ denotes the relative location of the child node in the parent grid layout at the m-th level, and all the operators here are entry-wise. Note that in Alg. 2 such localization calculation is done in a recursive way for function $FitIntoGrid$.

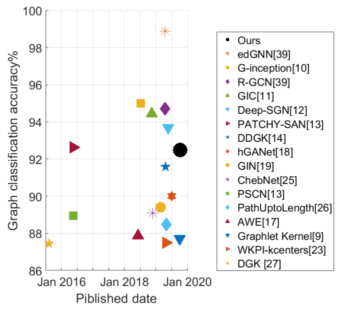
We illustrate this fitting procedure in the right side of Fig. 10.

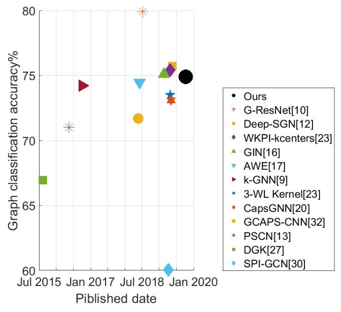
Constructing Undirected Connectivity Graph of Subgraphs. To build the connectivity graph $\mathcal{G}_H$ in Alg. 2, we first take each subgraph as one node in $\mathcal{G}_H$. Next we verify in the original graph whether there exists any edge that connects two subgraphs. If so, we add an edge between the two nodes in $\mathcal{G}_H$, otherwise, no edge.

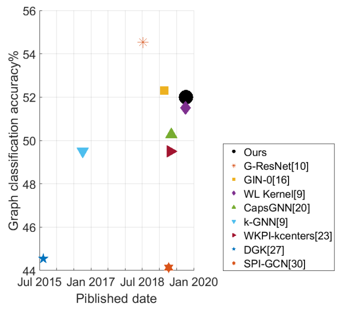
We illustrate this fitting procedure in the right side of Fig. 10. Firstly, we separated the 2048 points from the input point cloud into 32 clusters, and apply the GPGL algorithm to the cluster centers to create the high level layout. As shown in Fig. 10, the clusters labeled from 0 to 31 are projected to a 16×16 grid cell. For each cluster *e.g.* cluster 8, we apply the GPGL algorithm to the cluster nodes and generate a low level image representation with size 16×16, which is shown as the purple graph in the figure. Finally, we embedded the low level image into the cell in the high level grid, *e.g.* embedded the low level image of cluster 8 to the cell of cluster 8 in the high level grid. The grid cell without any cluster will be embedded with an empty 16×16 image. In that way we convert the 2048-node point cloud to a 256×256 image representation as illustrated on the right of Fig. 10.

4.2 Normalized Cuts [92] for Graph Partitioning

Improving the computational efficiency of GPGL is our concern in developing H-GPGL, where the $partition$ function is the key. Recall that the computational complexity of GPGL is (at least) proportional to the square of the number of node in the graph.


(a) MUTAG


(b) IMDB-B


(c) IMDB-M

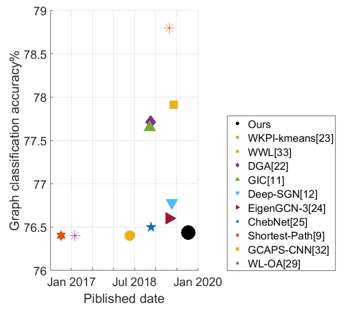

(d) PROTEINS

Figure 9: State-of-the-art result comparison. Numbers are cited from the top-20 of leaderboard at https://paperswithcode.com/task/graph-classification

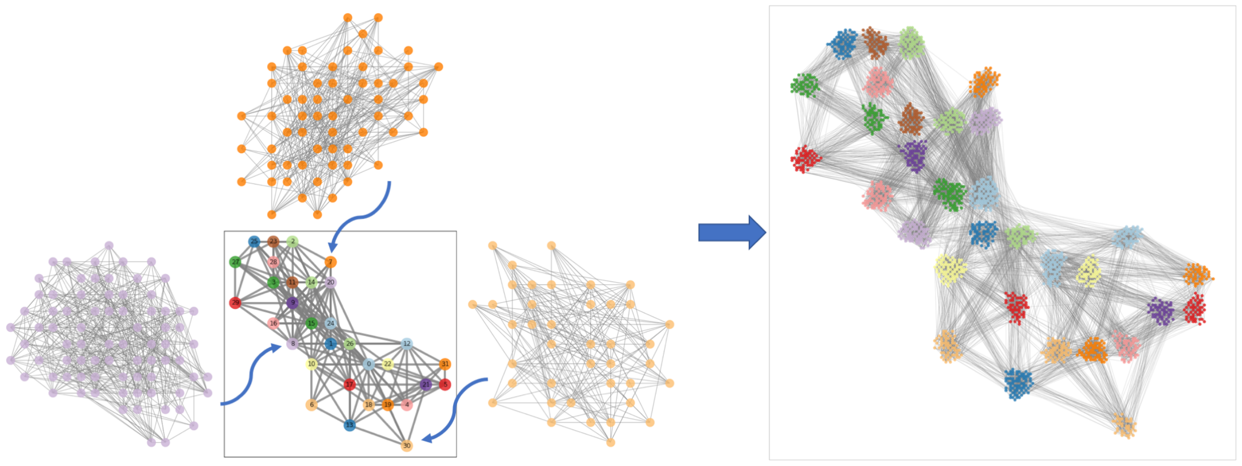


Figure 10: Illustration of two-level H-GPGL algorithm on a 2048-node graph, leading to a 256 × 256 grid layout.

Now given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the partition function should generate a set of J subgraphs $\{\mathcal{G}_j = (\mathcal{V}_j, \mathcal{E}_j)\}_{j=1,\cdots,J}$ that minimize the following problem:

$$\min_{\{\mathcal{G}_j\}} \sum_{j=1}^{J} \||\mathcal{V}_i|\|^2, \; s.t. \bigcup \mathcal{V}_j = \mathcal{V}. \tag{8}$$

The solution of the optimization problem above suggests that ideally the sizes of subgraphs should be equal.

This motivates us to employ the normalized graph cut algorithm as our partition function, because the algorithm aims to produce the subgraphs (given the number of subgraphs) with minimum cost as well as approximately equal sizes. Locating normalized cuts need to solve a generalized eigenvalue problem, whose computational complexity is essentially super-linear to the size of adjacency matrix of the graph [93]. Theoretically this complexity is no better than that of GPGL. Empirically, however, we do find that using normalized cut our H-GPGL performs much faster than GPGL alone. Fig. 11 illustrates our running speed comparison result. With a proper number of cuts like 16 or 32 in our case, H-GPGL can process the graph within 5.5s, while GPGL needs about 90s. With more cuts, the running time of normalized cut is longer and starting to dominate that of H-GPGL, leading to slower running speed.

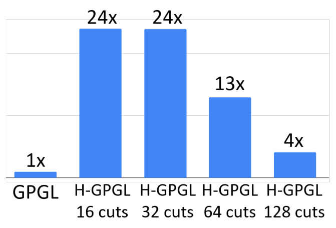


Figure 11: Running speed comparison on a 2048-node graph with two-level H-GPGL. See our experimental section later for more details.

Table 6: Running time (ms) comparison on a 2048 point cloud with 5-NN and Delaunay triangulation, followed by our H-GPGL.

Method	Graph construction	Normalized cut	GPGL	Total
5-NN	74.02	2036.37	3621.70	5732.09
Delaunay	1918.77	1703.07	1792.52	5414.36

4.3 Experiment: Point Cloud Semantic Segmentation

Consider a point cloud $\mathcal{P} \subseteq \mathbb{R}^3$ that contains a set of 3D points scanned from a single object. Each point $\mathbf{p} \in \mathcal{P}$ contains three geometry attributes x, y, z representing its location in the 3D space, and a part label $c \in \mathcal{C}$ that the point $\mathbf{p}$ belongs to. Fig. 13(a) shows a point cloud with 2048 points scanned from a skateboard. The problem of point cloud semantic segmentation is defined as predicting the part labels for all the 3D points in a point cloud.

In this section we will show how to apply our H-GPGL algorithm to the point cloud semantic segmentation problem. To do so, we first construct a graph for each point cloud, then map each graph onto a grid using H-GPGL, and finally utilize a variant of multi-scale maxout CNN to conduct the segmentation.

Data Set. For demonstration we use the ShapeNet [94] part segmentation benchmark, which contains 16881 object samples and each object sample has 2048 point scans associated to one of the 50 part categories. The benchmark is split into an 14007-object training set and a 2874-object testing set.

4.3.1 Graph Construction from Point Cloud

Graph construction from point cloud is the first step in graph-based approaches for point cloud applications. The graph in the literature is usually generated by connecting the K nearest neighbors (K-NN). However, the graph generation by K-NN often comes with difficulties in selecting a suitable K. On one hand, when K is too small, the points are intended to form small subgraphs (clusters) with no guarantee of connectivity among the subgraphs. This makes graph-convolution fail to pass through all graph nodes. On the other hand, when K is large, points are densely connected, leading to high processing load using graph kernels and large noise in local feature extraction.

In contrast to the K-NN based graph construction, our work employs the Delaunay triangulation [95], a widely-used triangulation method in computational geometry, to create graphs based on the positions of points. The triangulation graph has three advantages: (1) The connection of all the nodes in the graph is guaranteed, which makes graph-based network feasible on all constructed graphs from the point clouds; (2) All the local nodes are directly connected, which helps the local feature extraction; (3) The total number of graph connections is relatively low comparing to the graphs built by K-NN with a large K. Delaunay triangulation also returns better graphs that leads to better segmentation results (mcIoU & miIoU: 83.8%, 85.7%) than 5-NN (mcIoU & miIoU: 82.5%, 84.3%) using our MSM-CNN.

The worst-case computational complexity of Delaunay triangulation is well-known to be $O(n^{\lceil \frac{d}{2} \rceil})$ [96], which in the 3D space is $O(n^2)$. Table 6 lists the running time of graph construction algorithms followed by each key component in H-GPGL. Although Delaunay triangulation indeed needs significantly longer time, the overall running time is essentially better than 5-NN, because of better generated graphs.

4.3.2 Implementation Details

H-GPGL. The 2048 points in each object sample is first mapped to a graph using Delaunay triangulation. Then we conduct two-level H-GPGL to generate 32 subgraphs using normalized cut at the parent level whose connectivity graph is mapped to a 16×16 grid. Each subgraph is mapped to a 16×16 grid as well, leading to a 256×256 grid layout for each point cloud. Finally we generate 14007 training samples and 2874 test samples for further usage. Note that for this task we do not use data augmentation, due to the computational bottleneck.

CNN for Segmentation. In the experiment, we implement the following neural networks: (1) PointNet, (2) PointNet with 2D max-pooling and upsampling, (3) U-Net with VGG16 backbone, (4) U-Net with ResNet50 backbone, and (5) U-Net with MSM backbone. The architecture of PointNet with 2D max-pooing and upsampling is presented in Fig 12 The U-Net based networks are implemented using the segmentation-models package [97]. At the end of the network, a mask filter is applied to label the void pixels as ignored category, which do not participate in loss calculation and gradient back propagation.

We train the network for 30 epochs using Adam [91] with learning rate 0.0001 and batch-size 1. The training takes 15 hours for each neural network model on an NVidia 2080Ti GPU machine. We use two common metrics to evaluate the accuracy, mean instance intersect of union (miIoU) and mean class intersect of union (mcIoU).

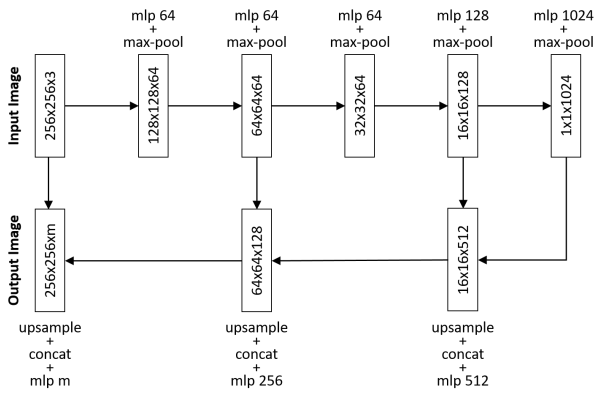


Figure 12: Architecture of PointNet with 2D max-pooling and upsampling for point cloud segmentation.

Segmentation on overlapped points. The H-GPGL on a point cloud may lead to part of the points overlapped in the same grid cell. Once overlapped points occur, we take an average, by default, of all the point and assign it to the grid cell. After network inference, the grid cell is labeled with a predicted semantic category, which is then assigned to all the overlapped points associated to that cell. In that way, we fuse the features of overlapped points and distribute the grid cell labels to those points.

4.3.3 Results

Table 7 presents the mcIoU and miIoU of all the neural network models on ShapeNet point cloud segmentation task. From the table we observe that the PointNet model achieves similar performance as in [64], while PointNet with 2D max-pooling and upsampling gains an improved performance with a large margin of 1.9% on mcIoU and 1.9% on miIoU. We also observe that all U-Net based models achieve an improved performance against the PointNet model, which indicates that 2D CNNs benefit from the GPGL image representations in point cloud feature learning.

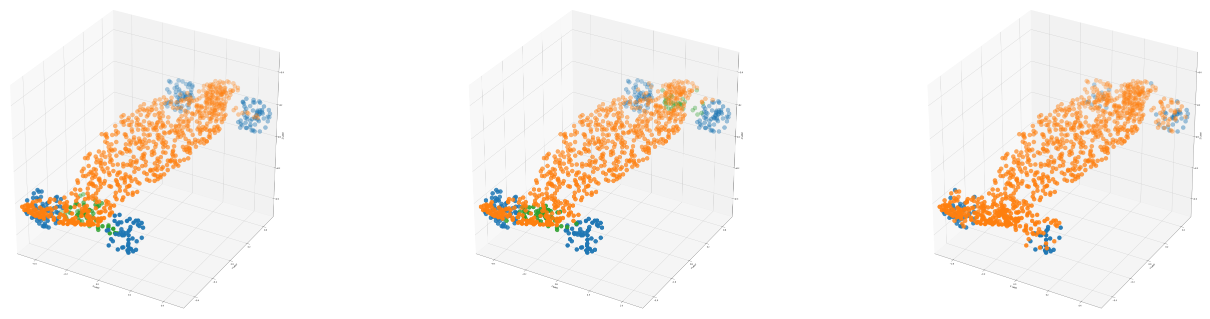
Table 7: Results of ShapeNet point cloud segmentation using different CNN models. The mcIoU denotes mean class intersect of union, and the miIoU denotes mean instance intersect of union.

Network	PointNet	PointNet with 2D pooling	VGG16	ResNet50	MSM
mcIoU	80.1	82.0	82.6	83.1	84.2
miIoU	80.4	82.3	82.8	84.0	86.0

We visualize our segmentation result in Fig. 13 comparing with the ground truth as well as the one using PointNet. As we see, our result is much closer to the ground truth, especially on the top-left wheel and the axis between the two bottom-wheels. Although we have some wrong predictions among the points between the two top-wheels, we think that these mistakes are acceptable because the shape is symmetric that makes the prediction harder to differentiate the top and bottom parts.

We then summarize the state-of-the-art segmentation results on ShapeNet in Fig. 14. Clearly our results using MSM-CNN, both mcIoU and miIoU, are comparable to the literature. Further we list the ShapeNet part segmentation IoU of each object class in Table 8 for more detail comparison, and ours are the best among 8 of the 16 classes.

Impact of overlapped points. To understand the impact of overlapped points in point cloud segmentation, we analysis the



(a) Ground Truth (b) Ours (c) PointNet

Figure 13: Visualization comparison of skateboard part segmentation results from **(a)** ground truth, **(b)** our H-GPGL, and **(c)** PointNet.

Table 8: Result comparison on ShapeNet part segmentation IoU (%) of each object class.

	air plane	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor bike	mug	pistol	rocket	skateboard	table	ave.
PointNet [64]	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6	80.4
Pointnet++ [65]	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6	81.9
DGCNN [79]	**84.2**	83.7	84.4	77.1	90.9	78.5	91.5	87.3	82.9	**96.0**	67.8	93.3	82.6	59.7	75.5	82.0	82.3
RS-CNN [69]	83.5	**84.8**	88.8	**79.6**	**91.2**	81.1	91.6	88.4	86.0	**96.0**	73.7	94.1	83.4	60.5	77.7	83.6	84.0
DensePoint [74]	84.0	85.4	**90.0**	79.2	91.1	**81.6**	91.5	87.5	84.7	95.9	**74.3**	**94.6**	82.9	64.6	76.8	83.7	**84.2**
Ours	83.3	83.1	89.4	75.9	87.8	80.5	**91.7**	**90.9**	**87.2**	**96.0**	66.6	92.6	**86.3**	**69.4**	**81.1**	**86.0**	**84.2**

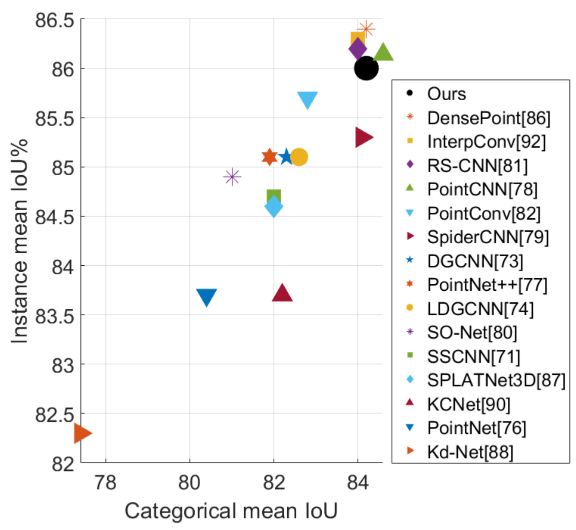


Figure 14: State-of-the-art segmentation result comparison on ShapeNet.

segmentation accuracy of all points and overlapped points on the ShapeNet test set. Among the 2874 test samples with 5,885,952 points, 17616 points are overlapped to the same grid cell, which covers 0.299% of the points. In the test, 62.7% of the overlapped points are mislabeled, which leads to a 0.22% decrease in mcIoU and 0.21% in miIoU. This observation shows that the MSM-CNN have difficulty in segmenting the overlapped points, which needs further study to overcome. However, since the overlapped points takes only tiny portion of the point cloud, they have minor impact to the accuracy of point cloud segmentation.

5 Conclusion

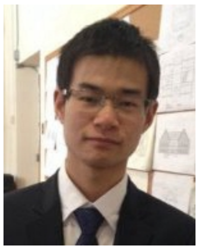
In this paper we answer the question positively that CNNs can be used directly for graph applications by projecting graphs onto grids properly. To this end, we propose a novel graph drawing problem, namely graph-preserving grid layout (GPGL), which is an integer programming to learn 2D grid layouts by minimizing topology loss. We propose a penalized Kamada-Kawai algorithm to solve the integer programming and a multi-scale maxout CNN to work with GPGL. We manage to demonstrate the success of GPGL on small graph classification. To improve the computational efficiency of GPGL, we propose hierarchical GPGL (H-GPGL) that utilizes graph partitioning algorithms such as normalized cut to generate subgraphs which GPGL is applied to. We demonstrate that H-GPGL is much suitable than GPGL to large graph applications such as 3D point cloud semantic segmentation, where we achieve the state-of-the-art on ShapeNet part segmentation benchmark. As future work we are interested in applying this method to real-world problems such as LiDAR data processing.

References

[1] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NeurIPS*, 2016, pp. 3844–3852.
[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[3] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.
[4] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.
[5] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *ICLR*, 2018.
[6] H. Gao and S. Ji, "Graph u-nets," in *ICML*, 2019, pp. 2083–2092.
[7] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.
[8] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *AAAI*, vol. 33, 2019, pp. 4602–4609.
[9] W. Zhao, C. Xu, Z. Cui, T. Zhang, J. Jiang, Z. Zhang, and J. Yang, "When work matters: Transforming classical network structures to graph cnn," *arXiv preprint arXiv:1807.02653*, 2018.
[10] J. Jiang, Z. Cui, C. Xu, and J. Yang, "Gaussian-induced convolution for graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4007–4014.
[11] Q. Xuan, J. Wang, M. Zhao, J. Yuan, C. Fu, Z. Ruan, and G. Chen, "Subgraph networks with application to structural feature space expansion," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[12] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *ICML*, 2016, pp. 2014–2023.
[13] R. Al-Rfou, B. Perozzi, and D. Zelle, "Ddgk: Learning graph representations for deep divergence graph kernels," in *The World Wide Web Conference*. ACM, 2019, pp. 37–48.
[14] H. Gao and S. Ji, "Graph representation learning via hard and channel-wise attention networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 741–749.
[15] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.
[16] S. Ivanov and E. Burnaev, "Anonymous walk embeddings," in *International conference on machine learning*. PMLR, 2018, pp. 2186–2195.
[17] F. Chen, S. Pan, J. Jiang, H. Huo, and G. Long, "Dagcn: Dual attention graph convolutional networks," *arXiv preprint arXiv:1904.02278*, 2019.
[18] Z. Xinyi and L. Chen, "Capsule graph neural network," in *ICLR*, 2019.
[19] Z. Zhang, D. Chen, J. Wang, L. Bai, and E. R. Hancock, "Quantum-based subgraph convolutional neural networks," *Pattern Recognition*, vol. 88, pp. 38–49, 2019.
[20] H. Jin, Q. Song, and X. Hu, "Discriminative graph autoencoder," in *2018 IEEE International Conference on Big Knowledge (ICBK)*. IEEE, 2018, pp. 192–199.
[21] Q. Zhao and Y. Wang, "Learning metrics for persistence-based summaries and applications for graph classification," *arXiv preprint arXiv:1904.12189*, 2019.
[22] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 723–731.
[23] B. Knyazev, X. Lin, M. R. Amer, and G. W. Taylor, "Spectral multigraph networks for discovering and fusing relationships in molecules," *arXiv preprint arXiv:1811.09595*, 2018.
[24] L. Jia, B. Gaüzère, and P. Honeine, "Graph kernels based on linear patterns: Theoretical and experimental comparisons," 2019.
[25] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.
[26] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: efficient graph kernels from propagated information," *Machine Learning*, vol. 102, no. 2, pp. 209–245, 2016.
[27] N. M. Kriege, P.-L. Giscard, and R. Wilson, "On valid optimal assignment kernels and applications to graph classification," in *NeurIPS*, 2016, pp. 1623–1631.
[28] A. Atamna, N. Sokolovska, and J.-C. Crivello, "Spi-gcn: A simple permutation-invariant graph convolutional network," 2019.
[29] S. Verma and Z.-L. Zhang, "Graph capsule convolutional neural networks," *arXiv preprint arXiv:1805.08090*, 2018.
[30] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt, "Wasserstein weisfeiler-lehman graph kernels," *arXiv preprint arXiv:1906.01277*, 2019.
[31] J. Li, Y. Rong, H. Cheng, H. Meng, W. Huang, and J. Huang, "Semi-supervised graph classification: A hierarchical graph perspective," in *The World Wide Web Conference*. ACM, 2019, pp. 972–982.
[32] R. Kondor and H. Pan, "The multiscale laplacian graph kernel," in *Advances in Neural Information Processing Systems*, 2016, pp. 2990–2998.
[33] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 22, no. 10, pp. 1533–1545, 2014.
[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[35] M. Chrobak and T. H. Payne, "A linear-time algorithm for drawing a planar graph on a grid," *Information Processing Letters*, vol. 54, no. 4, pp. 241–246, 1995.
[36] W. Schnyder, "Embedding planar graphs on the grid," in *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1990, pp. 138–148.
[37] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information processing letters*, vol. 31, no. 1, pp. 7–15, 1989.
[38] S. G. Kobourov, "Spring embedders and force directed graph drawing algorithms," *arXiv preprint arXiv:1201.3011*, 2012.
[39] Y. Frishman and A. Tal, "Multi-level graph layout on the gpu," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1310–1319, 2007.
[40] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.
[41] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 833–852, 2018.
[42] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.
[43] U. Doğrusöz, B. Madden, and P. Madden, "Circular layout in the graph layout toolkit," in *International Symposium on Graph Drawing*. Springer, 1996, pp. 92–100.
[44] M. Eiglsperger, S. P. Fekete, and G. W. Klau, "Orthogonal graph drawing," in *Drawing Graphs*. Springer, 2001, pp. 121–171.
[45] Y. Koren, "Drawing graphs by eigenvectors: theory and practice," *Computers & Mathematics with Applications*, vol. 49, no. 11-12, pp. 1867–1888, 2005.
[46] D. A. Spielman, "Spectral graph theory and its applications," in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. IEEE, 2007, pp. 29–38.
[47] R. Tamassia, *Handbook of graph drawing and visualization*. Chapman and Hall/CRC, 2013.
[48] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science*, vol. 290, no. 5500, pp. 2319–2323, 2000.
[49] R. Freese, "Automated lattice drawing," in *International Conference on Formal Concept Analysis*. Springer, 2004, pp. 112–127.
[50] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," vol. 97, pp. 2434–2444, 2019. [Online]. Available: http://proceedings.mlr.press/v97/grover19a.html
[51] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.
[52] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5708–5717.
[53] B. Samanta, D. Abir, G. Jana, P. K. Chattaraj, N. Ganguly, and M. G. Rodriguez, "Nevae: A deep generative model for molecular graphs," in *AAAI*, vol. 33, 2019, pp. 1110–1117.
[54] O.-H. Kwon and K.-L. Ma, "A deep generative model for graph layout," *IEEE Transactions on visualization and Computer Graphics*, vol. 26, no. 1, pp. 665–675, 2019.
[55] L. Franceschi, M. Niepert, M. Pontil, and X. He, "Learning discrete structures for graph neural networks," in *ICML*, 2019, pp. 1972–1982.
[56] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *CVPR*, 2017, pp. 5115–5124.
[57] Z. Huang, J. Wu, and L. Van Gool, "Building deep networks on grassmann manifolds," in *AAAI*, 2018.
[58] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *Proceedings of the IEEE international conference on computer vision workshops*, 2015, pp. 37–45.
[59] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein, "Learning shape correspondence with anisotropic convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 3189–3197.
[60] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Transactions on Signal Processing*, vol. 67, no. 1, pp. 97–109, 2018.
[61] L. Yi, H. Su, X. Guo, and L. J. Guibas, "Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation," in *CVPR*, 2017, pp. 2282–2290.
[62] K. Zhang, M. Hao, J. Wang, C. W. de Silva, and C. Fu, "Linked dynamic graph cnn: Learning on point cloud via linking hierarchical features," *arXiv preprint arXiv:1904.10014*, 2019.
[63] G. Te, W. Hu, A. Zheng, and Z. Guo, "Rgcnn: Regularized graph cnn for point cloud segmentation," in *2018 ACM Multimedia Conference on Multimedia Conference*. ACM, 2018, pp. 746–754.
[64] C. R. Qi, H. Su, K. Mo, and L. 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*, 2017, pp. 652–660.
[65] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in neural information processing systems*, 2017, pp. 5099–5108.

[66] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, “Pointcnn: Convolution on x-transformed points,” in *Advances in Neural Information Processing Systems*, 2018, pp. 820–830.
[67] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, “Spidercnn: Deep learning on point sets with parameterized convolutional filters,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 87–102.
[68] J. Li, B. M. Chen, and G. Hee Lee, “So-net: Self-organizing network for point cloud analysis,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9397–9406.
[69] Y. Liu, B. Fan, S. Xiang, and C. Pan, “Relation-shape convolutional neural network for point cloud analysis,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8895–8904.
[70] W. Wu, Z. Qi, and L. Fuxin, “Pointconv: Deep convolutional networks on 3d point clouds,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9621–9630.
[71] H. Lei, N. Akhtar, and A. Mian, “Octree guided cnn with spherical kernels for 3d point clouds,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9631–9640.
[72] A. Komarichev, Z. Zhong, and J. Hua, “A-cnn: Annularly convolutional neural networks on point clouds,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7421–7430.
[73] Z. Zhang, B.-S. Hua, and S.-K. Yeung, “Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1607–1616.
[74] Y. Liu, B. Fan, G. Meng, J. Lu, S. Xiang, and C. Pan, “Densepoint: Learning densely contextual representation for efficient point cloud processing,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5239–5248.
[75] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz, “Splatnet: Sparse lattice networks for point cloud processing,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2530–2539.
[76] R. Klokov and V. Lempitsky, “Escape from cells: Deep kd-networks for the recognition of 3d point cloud models,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 863–872.
[77] N. Verma, E. Boyer, and J. Verbeek, “Feastnet: Feature-steered graph convolutions for 3d shape analysis,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2598–2606.
[78] Y. Shen, C. Feng, Y. Yang, and D. Tian, “Mining point cloud local structures by kernel correlation and graph pooling,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4548–4557.
[79] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic graph cnn for learning on point clouds,” *Acm Transactions On Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.
[80] Y. Rao, J. Lu, and J. Zhou, “Spherical fractal convolutional neural networks for point cloud recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 452–460.
[81] J. Mao, X. Wang, and H. Li, “Interpolated convolutional networks for 3d point cloud understanding,” *arXiv preprint arXiv:1908.04512*, 2019.
[82] Y. Lyu, X. Huang, and Z. Zhang, “Learning to segment 3d point clouds in 2d image space,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 255–12 264.
[83] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *International Conference on Learning Representations*, 2015.
[84] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.
[85] A. Zisserman, “Self-supervised learning,” https://project.inria.fr/paiss/files/2018/07/zisserman-self-supervised.pdf, 2018.
[86] L. A. Wolsey and G. L. Nemhauser, *Integer and combinatorial optimization*. John Wiley & Sons, 2014.
[87] S. P. Bradley, A. C. Hax, and T. L. Magnanti, “Applied mathematical programming,” 1977.
[88] NetworkX developer team, “Networkx,” 2014. [Online]. Available: https://networkx.github.io/
[89] D. Harel and Y. Koren, “A fast multi-scale method for drawing large graphs,” in *International symposium on graph drawing*. Springer, 2000, pp. 183–196.
[90] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, “Maxout networks,” in *International conference on machine learning*, 2013, pp. 1319–1327.
[91] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2015. [Online]. Available: http://arxiv.org/abs/1412.6980
[92] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *Departmental Papers (CIS)*, p. 107, 2000.
[93] V. Vasudevan and M. Ramakrishna, “A hierarchical singular value decomposition algorithm for low rank matrices,” *CoRR*, vol. abs/1710.02812, 2017. [Online]. Available: http://arxiv.org/abs/1710.02812
[94] L. Yi, V. G. Kim, D. Ceylan, I. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, L. Guibas *et al.*, “A scalable active framework for region annotation in 3d shape collections,” *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, p. 210, 2016.
[95] B. Delaunay *et al.*, “Sur la sphere vide,” *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.
[96] N. Amenta, D. Attali, and O. Devillers, “Complexity of delaunay triangulation for points on lower-dimensional˜ polyhedra,” in *18th ACM-SIAM Symposium on Discrete Algorithms*, 2007, pp. 1106–1113.
[97] P. Yakubovskiy, “Segmentation models,” https://github.com/qubvel/segmentation_models, 2019.



Yecheng Lyu (S’17) received his B.S. degree from Wuhan University, China in 2012 and M.S. degree from Worcester Polytechnic Institute, USA in 2015 where he is currently a Ph.D student working on autonomous vehicles. His current research interest is point cloud processing and deep learning.

Xinming Huang (M’01–SM’09) received the Ph.D. degree in electrical engineering from Virginia Tech in 2001. Since 2006, he has been a faculty in the Department of Electrical and Computer Engineering at Worcester Polytechnic Institute (WPI), where he is currently a chair professor. Previously he was a Member of Technical Staffs with the Bell Labs of Lucent Technologies. His main research interests are in the areas of circuits and systems, with emphasis on autonomous vehicles, deep learning, IoT and wireless communications.

Ziming Zhang is an assistant professor at Worcester Polytechnic Institute (WPI). Before joining WPI he was a research scientist at Mitsubishi Electric Research Laboratories (MERL) in 2017-2019. Prior to that, he was a research assistant professor at Boston University in 2016-2017. Dr. Zhang received his PhD in 2013 from Oxford Brookes University, UK, under the supervision of Prof. Philip H. S. Torr. His research areas includes object recognition and detection, zero-shot learning, deep learning, optimization, large-scale information retrieval, visual surveillance, and medical imaging analysis. His works have appeared in TPAMI, CVPR, ICCV, ECCV, ACM Multimedia and NIPS. He won the R&D 100 Award 2018.