Group 1: A Survey of GNN on Point Cloud Segmentation

Hengda Shi
University of California, Los Angeles
United States
hengda.shi@cs.ucla.edu

Jiayue Sun
University of California, Los Angeles
United States
jysun@cs.ucla.edu

Benlin Liu
University of California, Los Angeles
United States
liubenlin@cs.ucla.edu

ABSTRACT

This survey discusses a set of graph convolutional neural network models on point cloud segmentation task. Point cloud segmentation has always been a challenging task as the data size could grow exponentially. Compared to the traditional methods like multi-view method (performing 2D convolution on a set of 2D snapshot from the point cloud), voxel method (transform the point cloud into voxel), consuming raw points from the point cloud has significant benefits in learning the characteristics directly on the point cloud structure. In this survey, we will discuss the point-based methods, the dynamic graph-based methods, the hierarchical-based methods, and the scalable GCN methods. The evaluation and comparison will be discussed in the benchmark and evaluation section.

KEYWORDS

deep learning, graph neural networks, point clouds, part segmentation, semantic segmentation

ACM Reference Format:
Hengda Shi, Jiayue Sun, and Benlin Liu. 2021. Group 1: A Survey of GNN on Point Cloud Segmentation. In *CS249 Group 1.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Point cloud has been widely studied in recent years. It can be easily obtained by using 3D scanners like RGB-D camera. The captured point cloud can reflect the struture of the object or the whole scene, and is very different from CNN inputs defined on regular grids with uniform constant density. Traditional methods either project the point clouds to multiple planes and use CNN to process the projection, or partition the point cloud to the 3D voxels and use 3D CNN model to learn from the data. However, the former may lead to the missing of structural information caused by projection, while the latter may have the problem that some voxels are just too empty as the point clouds are very sparse. Due to these problems, it is very alluring to learn directly from point cloud. PointNet [12] is the first attempt in this direction. In particular, PointNet uses a MLP style model to process each point independently, and thus lack the ability to capture the local structure of the point cloud. To improve on this, PointNet++ proposes to learn the feature from point cloud in a hierarchical way like CNN, which involves sampling, grouping and aggregation operation. And this can be replaced by incorporating graph neural networks. So in this survey, we study the use of GNN for point colud processing.

CS249 Group 1, Mar 19, 2021, LA, CA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 MODELS

In this survey, two classical models - PointNet and PointNet++ that directly consumes raw points from the point cloud will be discussed first to give a baseline of the survey. The graph convolutional neural networks are then divided into three subcategories. The first category is the dynamic graph based methods which reconstructs the graph dynamically at each hidden layer to obtain a larger receptive field, and perform graph convolutions on the constructed graph to aggregate neighboring information. The second category which is the hierarchical based methods follows the classic encoder-decoder architecture in PointNet++ and 2D image segmentation tasks while subsampling and interpolating the graph using certain methods. The third category, the scalable GCN methods, attempts to find graph convolution and subsampling operations that can effectively and efficiently process point cloud data.

2.1 Point-based Methods

2.1.1 *PointNet*

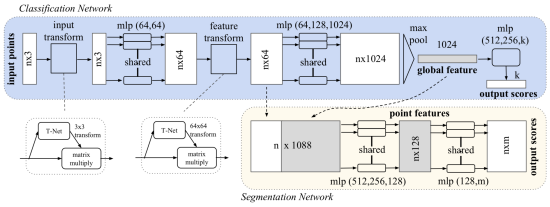

Figure 1: The architecture of PointNet.

PointNet[12] only captures global structure but not local structure. We briefly introduce PointNet to indicate how to use PointNet to aggregate the point features.

As shown in Fig. 1, PointNet applies MLP to process points independently. In particular, PointNet first use a STN-like module to align the point cloud to a canonical space, then use a MLP to increase the size of the point embedding. To be invariant to the permutation, PointNet uses max pooling operation to aggregate the point features. For segmentation task, we can concatenate the learned global feature to the point features to capture the local structure.

The PointNet can be used to aggregate any amount of point features in the following literature. Many following works attempt to better capture the local structure by applying more sophisticated

sampling and grouping strategy, which can be related to GNN research.

2.1.2 *PointNet++*

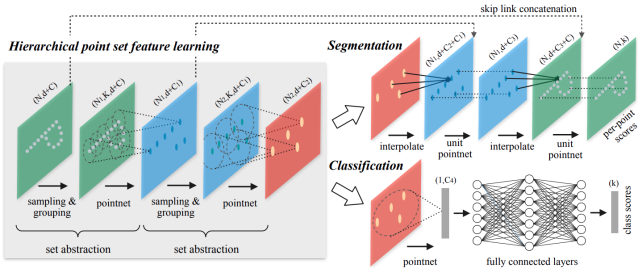

Figure 2: The architecture of PointNet++

By its design, PointNet only captures global graph structure and lacks the ability to capture local context. Yet considering the success of convolutional architecture models on segmentation tasks through progressively extract local patterns of increasingly large receptive fields along a multi-resolution hierarchy, PointNet++[13] proposes to add hierarchical structure to the original PointNet. It leverages neighborhoods at multiple scales to achieve both robustness and detail capture. To be specific, when it is applied to a segmentation task, PointNet++ will have an encoding and a decoding stage. In each level of the encoding stage, input points are abstracted into fewer points whose features are determined by both their local region features and the global features. Such abstraction consists of three key layers: sampling, grouping and feature extraction.

For the sampling layer, it uses iterative farthest point sampling (FPS) to choose a subset of points as centroids of local regions. Basically, it iteratively samples the point farthest away from the collected set among the rest points until the requested number of sample points is reached. Then the grouping layer constructs local region by grouping maximum K neighboring points within a radius to the sampled centroids. With the local point sets established, PointNet is used to extracts local pattern features. The input to PointNet are local regions represented by their relative point positions to the centroid $x_i^{(j)} - \hat{x}^{(j)}$ along with their point features. Then the PointNet can output a feature that represents this local region.

Since the segmentation task requires labels for all the points, the model eventually needs to output point features for all the original points. Thus, after the points got downsampled in the encoding stage, PointNet++ has another hierarchical decoding stage that propagates point features from the last layer in the encoder stage to all the original input points. In each decoding layer, point features of those not existing in the previous layer are computed through interpolating features from their corresponding point sets in the previous layer. The interpolation is the average of kNN features weighted by their inverse point distances, which is specified in Eq 1. Then the interpolated features are concatenated with skip linked point features from the encoding stage. These skip connections are between encoding and decoding layers with the same number of points, as shown in Figure 2. Then the new point features are obtained through passing these concatenated features through a unit PointNet and a few fully connected layers.

$$f^{(j)}(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i^{(j)}}{\sum_{i=1}^{k} w_i(x)} \text{ where } w_i(x) = \frac{1}{d(x, x_i)^p}, j = 1, ..., C \tag{1}$$

2.2 Dynamic Graph based Methods

In the section of dynamic graph convolutional neural networks, we aim to review a set of papers that dynamically updates the graph at each hidden layers. The point cloud is a set of points with coordinate information and initially does not have any edge connections.

Many proposed methods construct the graph in a certain way to allow the information flows through the edges. For a point cloud that contains n points, the set of features with size d for each point including the geometric, color, and other information like surface norm can be denoted as $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\} \in \mathbb{R}^{n \times d}$. A **directed** graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed where $\mathcal{V} = \{1, 2, \ldots, n\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$.

A generalized convolution operation has been defined in Wang et al. [16], and we will follow the notation in this survey. Specifically, a convolutional operation at the i-th vertex is defined as

$$\boldsymbol{x}_i' = \underset{j:(i,j)\in\mathcal{E}}{g} h_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2}$$

where $h_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the edge feature function between the central vertex $\boldsymbol{x}_i$ and one of the neighboring vertex $\boldsymbol{x}_j$ with a set of learnable parameters $\boldsymbol{\Theta}$, and g is the aggregation function (e.g., $\sum$ or max).

The PointNet structure can be interpreted as using the edge feature function

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = h_{\boldsymbol{\Theta}}(\boldsymbol{x}_i) \tag{3}$$

where the edge feature function h only considers the central vertex but not any neighboring vertices. The ignorance of the neighboring features becomes one of the main drawback of PointNet and the later work PointNet++ and other models try to improve it with graph convolutional neural network.

In PointNet++, the edge function can be written as

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = h_{\boldsymbol{\Theta}}(\boldsymbol{x}_j) \tag{4}$$

where the edge function only learns a relative information with respect to the central point. The central point information is neglected in the edge function making it less informative compared to the later approaches.

For each model in the subsections, we will introduce the graph construction method and the convolution method using the above formulation. The experiment results will be shown in the next section.

2.2.1 *DGCNN*

Dynamic Graph CNN [16] is one of the recent popular methods in applying graph convolution operation on point cloud. Inspired by recent work like PointNet [12] and GCN [1, 8], DGCNN constructs local graph in the point cloud and uses graph convolution operation called *EdgeConv* to perform convolution operation on these local graphs.

In particular, DGCNN constructs the directed graph $\mathcal{G}$ by performing k-nearest neighbor algorithm at each layer. Such dymanic

graph update is beneficial for the model to learn a larger receptive field because the nearest neighbor is likely to change after each hidden layer. Formally, a different graph $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$ will be constructed at the l-th layer where $\boldsymbol{x}_{j_{i1}}^{(l)}, \ldots, \boldsymbol{x}_{j_{ik_l}}$ are the k_l points that are closest to $x_i^{(l)}$ based on a distance metric.

In designing the local graph convolution operation, the drawback of only incorporating central vertex or neighboring vertices is identified in this paper. To incorporate both information, a edge feature function that incorporates central vertex information and relative information from neighboring vertices is developed

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \bar{h}_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j - \boldsymbol{x}_x) \tag{5}$$

The entire *EdgeConv* can be denoted as

$$\boldsymbol{x}'_{im} = \max_{j:(i,j)\in\mathcal{E}} \text{ReLU}(\boldsymbol{\theta}_m \cdot (\boldsymbol{x}_j - \boldsymbol{x}_i) \oplus \boldsymbol{\phi}_m \cdot \boldsymbol{x}_i) \tag{6}$$

where $\boldsymbol{\Theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M, \boldsymbol{\phi}_1, \ldots, \boldsymbol{\theta}_M)$, and $\oplus$ denotes a concatenation.

An important note on this choice of function is that it is asymmetric. It leads to the problem of translation variance meaning that the *EdgeConv* proposed in this work could produce different output once the input is transformed with rotation operation or other transformation. The author proved their *EdgeConv* to be partially translation invariant. By applying a shifting factor T, the output of the *EdgeConv* is partially preserved

$$\boldsymbol{x}'_{im} = \max_{j:(i,j)\in\mathcal{E}} \text{ReLU}(\boldsymbol{\theta}_m \cdot ((\boldsymbol{x}_j + T) - (\boldsymbol{x}_i + T)) \oplus \boldsymbol{\phi}_m \cdot (\boldsymbol{x}_i + T))$$

$$\boldsymbol{x}'_{im} = \max_{j:(i,j)\in\mathcal{E}} \text{ReLU}(\boldsymbol{\theta}_m \cdot (\boldsymbol{x}_j - \boldsymbol{x}_i) \oplus \boldsymbol{\phi}_m \cdot (\boldsymbol{x}_i + T))$$

However, it is still technically translation variant and the choice of the *EdgeConv* breaks the nice property of the point cloud.

DGCNN closely follows the PointNet setup to mitigate the translation variance issue by performing a spatial transformation at the start of the network. However, it is shown in the later work that the memory burden brought by the spatial transformation outweighs the performance benefit.

In terms of network design, they aggregate all *EdgeConv* outputs at the last stage before the global MLP to incorporate all layers information. The design of the skip connection will also be explored in later works.

2.2.2 *LDGCNN*

Linked DGCNN [17] is a close extension of DGCNN [16]. In LDGCNN, they adopts the graph construction and *EdgeConv* methods, but instead makes a few modification based on DGCNN.

Specifically, the spatial transformation step is removed as it nearly doubles the memory usage. They claims that the MLP in the network can approximate the spatial transformation by comparing the offsetting operation and the MLP layer operation. To further help the network adopt translation invariance, they instead transfer the overhead to preprocessing by augmenting the training data with operations similar to PointNet++ [13].

Another contribution in LDGCNN is to change the design of the skip connection. The original skip connection is to simply aggregate all layer outputs to the global MLP layer. However, such connection design does not help the *EdgeConv* layers to incorporate information from earlier layers. To mitigate the issue, LDGCCN adopts the skip connection design from DenseNet [6], i.e. to connect the earlier layer output with all later layer outputs. In terms of the model performance, the overall accuracy improvement is negligible. However, the model size is shrunk by twice and therefore it allows the model to more efficiently perform training and inference.

2.2.3 *DeepGCN*

DeepGCN [9] extensively studies the effect of the different skip connection design. Other than the skip connection design adopted by DGCNN (denoted as PlainGCN) and LDGCNN (denoted as DenseGCN), DeepGCN adopts the skip connection design used in ResNet [3]. A skip connection comparison graph is given in figure 3.

By investigating the skip connection design, DeepGCN aims to stack more layers to allow the model learn more complex representation. Extensive ablation study has been done by using different convolution layers, and the result shows that the *EdgeConv* proposed in DGCNN performs the best.

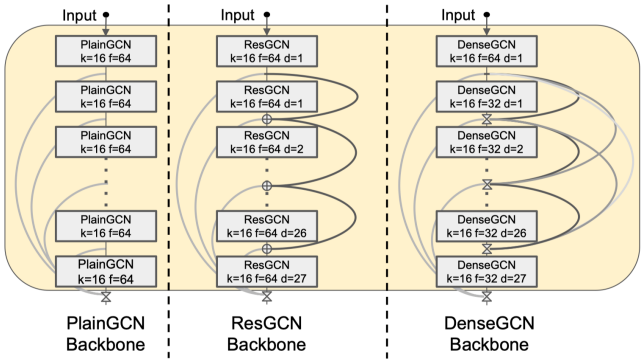

Figure 3: The architecture of DeepGCN

In addition to the experiment of network skip connection design, DeepGCN also modified the graph construction to help the model learn a larger receptive field. The original k-nearest neighbor algorithm densely collects the closest k neighbors based on the distance metrics. However, such method has a limited receptive field so that it can only learn the local information within the receptive field. DeepGCN proposes to use *Dilated k-NN* to construct a *Dilated Graph* at each layer's graph reconstruction. Formally, for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with dilated rate p, if $(\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_{k\times p})$ denotes the first $k \times p$ neighbors, then the dilated k-nearest neighbors would be $(\boldsymbol{u}_1, \boldsymbol{u}_{1+p}, \ldots, \boldsymbol{u}_{1+(k-1)p})$. By skipping neighbors at a fixed rate, the model is capable of aggregating more information for each vertex.

With the modification of the *skip connection design*, and the *dilated k-NN* at each layer's graph reconstruction, DeepGCN with a ResGCN backbone can stack 28 GCN layers with 4% performance improvement in mIoU over DGCNN on S3DIS.

2.2.4 *RGCNN*

Regularized GCNN [14] still adopts the dynamic graph construction setting, but uses a different graph construction scheme as well as the graph convolution operation.

In the graph construction stage, RGCNN constructs a complete graph by connecting each vertex with all other vertices in the point

cloud, and assign the weight $e_{i,j}$ on each edge based on the distance metric

$$e_{i,j} = \exp(-\beta \|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2^2) \tag{7}$$

where β is a hyperparameter and usually set to $\beta = 1$ in their experiments.

The benefit of using such graph construction is to largely mitigate the cost of k-NN at each layer. However, it comes with the cost of high memory usage because the adjacency matrix is usually stored as a sparse matrix but with a complete graph it has to be a dense matrix.

RGCNN adopts the graph convolution operation *ChebConv* proposed in ChebNet [1]. *ChebConv* defines the convolution operation from the spectral domain trying to aggregate neighboring information up to K-th hop. Following the notation from DGCNN, the *ChebConv* is defined as

$$\boldsymbol{x}_i' = \text{ReLU}(g_\theta(\mathcal{L})\boldsymbol{x}_i\boldsymbol{W}_i + \boldsymbol{b}_i) \tag{8}$$

and $g_\theta(\mathcal{L})\boldsymbol{x}_i$ is defined as

$$g_\theta(\mathcal{L})\boldsymbol{x}_i = \sum_{k=0}^{K-1} \theta_k \cdot T_k(\mathcal{L}) \cdot \boldsymbol{x}_i \tag{9}$$

where θ_k is the k-th Chebyshev coefficient in the truncated Chebyshev polynomials, and $T_k(\mathcal{L})$ is the Chebyshev polynomial of order k. The k-th order Chebyshev polynomial is recurrently computed as

$$T_k(\mathcal{L}) = 2\mathcal{L}T_{k-1}(\mathcal{L}) - T_{k-2}(\mathcal{L}) \tag{10}$$

where $T_0(\mathcal{L}) = 1$ and $T_1(\mathcal{L}) = \mathcal{L}$. $\mathcal{L}$ denotes the *normalized* Laplacian matrix, i.e. if the original adjacency matrix is denoted as $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{D} \in R^{n\times n}$ is the degree diagonal matrix with $d_{i,i} = \sum_{j=1}^{n} a_{i,j}$, and $\mathcal{L}_c = \boldsymbol{D} - \boldsymbol{A}$ denotes the Laplacian matrix, then the *normalized* Laplacian matrix is

$$\mathcal{L} = \boldsymbol{D}^{-\frac{1}{2}}\mathcal{L}_c\boldsymbol{D}^{-\frac{1}{2}} \tag{11}$$

Although the *ChebConv* seems to be complex at the first glance, it is essentially gathering vertex information up to K-th hop. With the localized benefit, RGCNN experimented $K = (6, 5, 3)$ to gather information far from the central vertex.

To further impose similarity constraints on nearby vertices, RGCNN proposes a smoothness prior to be added to the loss. The graph $\mathcal{G}$ will be smoothed if

$$\sum_{i\sim j} e_{i,j}(y_i - y_j)^2 < \epsilon,\ \forall i, j \tag{12}$$

where y_i, y_j denotes the final outcome for vertices i and j, ϵ is a positive scalar to tune, and $i \sim j$ denotes the one-hop neighbors. By observing the constraint, the difference between y_i and y_j shall be smaller as the edge weight $e_{i,j}$ becomes larger. The smoothness prior is then

$$\gamma \sum_{l=0}^{2} \boldsymbol{y}_l^\top \cdot \mathcal{L} \cdot \boldsymbol{y}_l = \gamma \sum_{l=0}^{2} \sum_{i\sim j} e_{i,j}^{(l)} (y_i^{(l)} - y_j^{(l)})^2 \tag{13}$$

where γ is a penalty parameter that is set to 10^{-9} in the experiments.

In terms of the network design, RGCNN barely uses the skip connection in the model architecture except they draw one skip connection from one of the *ChebConv* output to the segmentation classifier. This design choice could potentially be the cause of the negligible accuracy improvement. The graph construction choice leads to a tradeoff in space and time complexity. The forwarding time is much faster compared to other models listed in table 3. However, the model size is nearly the same as DGCNN whose size can be reduced by half as demonstrated in LDGCNN.

Method	Model Size (MB)	Forward Time (ms)
PointNet	40	25.3
PointNet++	**12**	163.2
DGCNN	21	94.6
RGCNN	22.4	**7.5**

Table 1: Complexity Comparison

2.3 Hierarchical based Methods

For hierarchical based methods, the models generally follow the encoder-decoder architecture in PointNet++. Encoder-decoder architecture has been effective in image object detection and segmentation tasks. Due to the large amount of points in the graph, subsampling is another common technique that most of the models will deploy in their model. The choice of the subsampling technique could lead to a noticeable performance or efficiency gain. In the next few subsections, we will introduce a set of models that adopt and modify some aspects of the classic model PointNet++.

2.3.1 *GACNet*

Graph-Attention-Convolution Net [15] adopts the encode-decode architecture used in PointNet++. As the name *graph attention* suggests, GACNet utilizes the attention mechanism to selectively aggregate neighboring vertices information.

In the graph construction stage, GACNet chose to randomly sample k neighbors within the radius r. Similar to PointNet++, they adopt the furthest point sampling (FPS) to subsample points before passing into the neural network. The graph reconstruction and subsampling technique is expensive to perform at each layer's update, so GACNet chose to alternatively apply each of the techniques at the convolutional layers. Hence, GACNet is also a partially dynamic graph CNN model as it does dynamically update the graph every few layers. In addition, they perform principle component analysis on the k-nearest neighbors and take the eigenvalues as addition features.

The sharing attention mechanism $\alpha : \mathbb{R}^{D_{l-1}} \to \mathbb{R}^{D_l}$ where D_{l-1}, D_l are the dimensions of the last layer and the current layer respectively. The attention weight $\tilde{\boldsymbol{a}}_{ij}$ for each pair of vertices with geometric information $p_i, p_j \in \mathbb{R}^3$ and the rest of the feature vector $q_i, q_j \in \mathbb{R}^{D_{l-1}-3}$ is then computed as follows

$$\tilde{\boldsymbol{a}}_{ij} = \alpha(\Delta p_{ij}, \Delta q_{ij}),\ j \in \mathcal{N}(i) \tag{14}$$

$$= \text{MLP}_\alpha([\Delta p_{ij} \oplus \Delta q_{ij}]) \tag{15}$$

where $\tilde{\boldsymbol{a}}_{ij} = \{\tilde{a}_{ij,1}, \tilde{a}_{ij,2}, \ldots, \tilde{a}_{ij,D_l}\} \in \mathbb{R}^{d_l}$, $\Delta p_{ij} = p_i - p_j$, $\Delta q_{ij} = \text{MLP}_h(q_i) - \text{MLP}_h(q_j)$, and $\oplus$ indicates a concatenation. $\mathcal{N}(i) = \{j : (i, j) \in \mathcal{E}\} \cup \{i\}$ Δp_{ij} helps the model to map the simliar vertices into close subspace, and Δq_{ij} helps the model to capture the feature difference. An additional MLP_h is used in feature difference

computation to give the model more flexibility in modeling the attention weights.

Due to the size-varying neighbors, the attention is further normalized across all neighbor vertices with respect to vertex i

$$\tilde{\boldsymbol{a}}_{ij,d} = \frac{\exp(\tilde{\boldsymbol{a}}_{ij,d})}{\sum_{l \in \mathcal{N}(i)} \exp(\tilde{\boldsymbol{a}}_{ij,d})} \tag{16}$$

An illustrated graph of how the attention is applied is shown in figure 4.

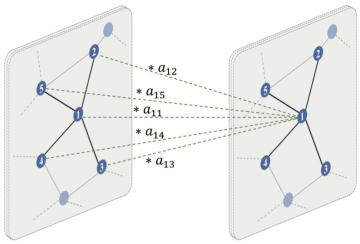

Figure 4: The attention mechanism in GACNet

In terms of network design, GACNet adopts the general encoder-decoder architecture from Lin et al. [11], Qi et al. [13]. The skip connection is drawn for the layers at the same scale. This type of architecture is proven to be effective at capturing information in the data at different scale. At the decoder stage, the features are interpolated using the k-NN interpolation. Specifically, if p_l and p_{l-1} are the spatial coordinates for the same point in the l-th layer and $l-1$-th layer respectively, three nearest neighbors of p_{l-1} at p_l will be obtained and a weighted sum of the features will be served as the interpolated feature.

2.3.2 *HDGCN*

HDGCN follows the encode-decode hierarchical structure of PointNet++. It also uses FPS to downsample points but uses a depthwise graph convolution block (DGConv block) instead of PointNet as local feature learner.

Similar to how MobileNet[4] breaks down normal convolution layer into two parts, depthwise convolution and pointwise convolution, HDGCN also breaks down graph convolution to depthwise graph convolution and pointwise convolution to reduce the memory and computation cost. Specifically, it applies graph convolution to channels independently first and then apply pointwise 1×1 convolution (MLP) to the generated channel features, as shown in Figure 5.

For a specific point x_i and its kNN points $\{x_{j_{i_1}}, \ldots, x_{j_{i_k}}\}$, they are connected by edges $L(i, j_{i_m}) = p_{j_{i_m}} - p_i (m = 1, \ldots, k)$. The input point feature of graph convolution is $x_i \in \mathbb{R}^I$ and the output feature is $x'_i \in \mathbb{R}^O$. Then the spatial graph convolution should be formulated as:

$$x'_i = \sum_{m=1}^{k} MLP(L(i, j_{i_m}))x_{j_{i_m}} \tag{17}$$

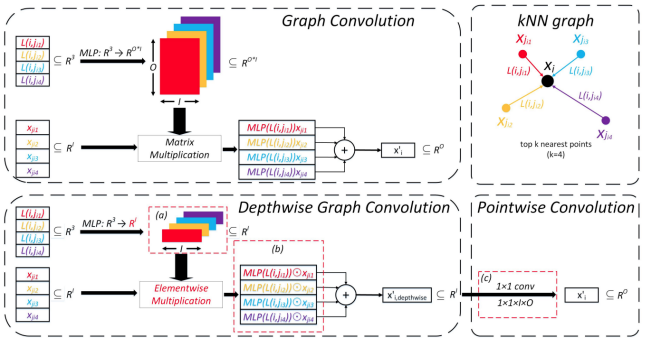

Figure 5: Depthwise Graph Convolution vs. spatial graph convolution.

HDGCN breaks down the above graph convolution into depthwise graph convolution and pointwise convolution that are formulated as follows:

$$\begin{aligned} x'_{i,depthwise} &= \sum_{m=1}^{k} MLP(L(i, j_{i_m}))x_{j_{i_m}} \\ x'_i &= MLP(x'_{i,depthwise}) \end{aligned} \tag{18}$$

Here the dimension of depthwise graph convolution output is $x'_{i,depthwise} \in \mathbb{R}^I$, and thus the pointwise convolution MLP maps features from $\mathbb{R}^I$ to $\mathbb{R}^O$. We know that the memory demand of MLP in Eq 17 is $(n \times k \times O \times I)$, and the memory demand of the graph convolution MLP is $(n \times k \times 2I + I \times O)$. When the ouput feature dimension O is large, the memory needed for the depthwise version is much smaller.

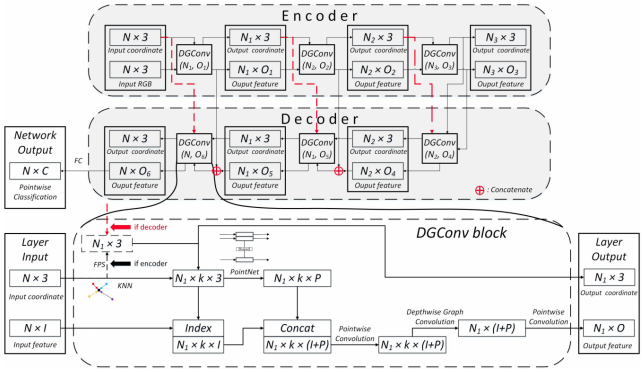

Figure 6: The architecture of HDGCN and its DGConv Block

Another contribution of HDGCN [10] is that they construct a new convolutional feature extraction module DGConv. HDGCN comprises of a series of DGConv unit blocks. When encoding, DGConv blocks are used to iteratively downsample points and extract features. When decoding, skip connection is used to provide upsampled point coordinates and concatenate features from the corresponding encoder layers.

You can see in Figure 6, for each sampled point, DGConv block uses kNN to construct the local region graph and a small PointNet is used to extract point features from this local graph of point

coordinates. This spatial encoding part is very similar to that of PointNet++ and the difference is that DGConv also aggregate these extracted point features with the indexed input features from previous block output. Then the result of this aggregation got passed into the depthwise graph convolution to generate the output features of DGConv block.

The main contributions of HDGCN [10] are that it proposes a more memory efficient convolution to aggregate local features, and that their proposed DGConv is effective in extracting both local and global features of point clouds.

2.3.3 *HPEIN*

Hierarchical Point-Edge Interaction Network (HPEIN) [7], follows the hierarchical enoder-decoder structure of PointNet++ that have skip-connections linking layers with matching size in the encoder and decoder. Yet unlike previous methods that primarily focuses on point features, Hierarchical Point-Edge Interaction Network (HPEIN) introduces a new edge branch into the decoder that interacts with the point branch and is also supervised by a loss. The encoder of HPEIN remains the same as that in PointNet++, an iteration of point downsampling and extracting local features.

This edge branch in the decoder composed of consecutive edge modules that can progressively integrate point features in different layers. It takes both features from the corresponding point module and the previous edge module. The procedure is to extract edge features from the coarsest layer to grab high-level information with the largest receptive field, and progressively fuse point features from finer layers into edges, in parallel with the point decoding stage.

Each edge module takes in L-layer point features $\mathbb{F}_{V_L}$ and $(L-1)$-layer edge features $\mathbb{H}_{E_{L-1}}$ as arguments and returns the edge features in layer L. This module first upsamples the edge features from the previous layer and then encodes it along with the point features at the current layer. It can be expressed as:

$$\mathbb{H}_{E_L} = M_{encoder}(\mathbb{F}_{V_L}, M_{upsample}(\mathbb{H}_{E_{L-1}})) \tag{19}$$

Let's first take a look at how the edge feature upsampling $M_{encoder}$ is accomplished. Most previous methods like PointNet++ and HDGCN do not have edge features and their "edges" are simply the relative spatial position of neighbor points to the centroids in the local graph. Thus when upsampling points in the decoding stage, they only need to upsample points and the edges will be computed based on the upsampled points. Yet since edges in HPEIN have their own features, they also need to be upsampled in the decoding stage. Hence, HPEIN proposes a way to hierarchicaly construct graphs for the upsampled points and encode them with kNN interpolation.

Consider two adjacent layers $L-1$ and L with vertices V_{L-1} and V_L as the point set in that layer, respectively. The graph G_L is constructed by first finding the kNN points for each point in V_L and we get the inital L-layer graph $G_L^{(0)} = (V_L, E_L^{(0)})$. For each edge $e_{i,j} = (p_i, p_j)$ in the initial edge set $E_L^{(0)}$, HPEIN finds existing edges between the kNN neighbors of p_i and p_j in layer $L-1$ and denotes them as the nearest edges for edge $e_{i,j}$. Then it constructs the final L-layer edge set E_L by taking the union of nearest edges of all edges in the kNN neighborhoods. The final constructed graph is $G_L = (V_L, E_L)$.

Then similar to how the point features got propagated in the decoding stage of PointNet++ using kNN interpolation(Eq 1), HPEIN also propagtes the edges features from layer $L-1$ to layer L by interpolating the nearest edges of an edge in layer $L-1$, and it also uses inverse distance of the two pairs of end points for edge (p_i, p_j) in layer L and edge $(p_{i'}^{'}, p_{j}^{'})$ in layer $L-1$. The interpolation and the inverse distance weight is formualted as:

$$H_{i,j}^{L-1\to L} = f_{interp}^{e}\left(\left\{H_{i',j'}^{L-1\to L} \mid (p_{i'}, p_{j'}) \in E_{ne}^{L-1}\left(e_{i,j}\right) \cap E_{L-1}\right\}\right) \tag{20}$$

After obtaining the upsample edge feature $H_{i,j}^{L-1\to L}$ of edge $e_{i,j} = (p_i, p_j)$ from layer $L-1$ to layer L, we need to use $M_{encoder}$ to encode this edge feature along with point features of p_i and p_j, F_i^L and F_j^L. The feature extractor f_{ext} is MLP and the edge function f_{edge} is a concatenation of relative point position and point features pf the two end points.

$$H_{i,j}^{L} = f_{ext}^{(1)}\left(\left[f_{ext}^{(2)}\left(f_{edge}\left(F_i^L, F_j^L\right)\right), H_{i,j}^{L-1\to L}\right]\right) \tag{21}$$

$$f_{edge}\left(F_i^L, F_j^L\right) = \left[\left(p_i - p_j\right), F_i^L, F_j^L\right] \tag{22}$$

After getting the encoded edge features $\mathbb{H}_{E_L}$, they are aggregated and be used to update the point features so that the edge contextual information can be passed back to the point. Then the point feature F_i^L can be updated by

$$\left(F_i^L\right)_{new} = \left[F_i^L, \text{MaxPool}\left(\mathbb{H}_{E_L(p_i)}\right)\right] \tag{23}$$

Eventually, the final layer of edge features are supervised by semantic consistency of related points. The label for edge $e_{i,j} = (p_i, p_j)$ is set as:

$$l_{i,j}^{e} = \begin{cases} 1, \text{if } l_i^p = l_j^p \\ 0, \text{if } l_i^p \neq l_j^p \end{cases}$$

This edge loss guides the edge encoder to seek difference between the intra- and inter-class feature pairs, and implicitly serves as auxiliary supervision for point features. It increases the discrimination power among point features in different categories.

The main contribution of HPEIN is that it proposes a new edge branch that hierarchically encodes edge features and interacts with the point branch. This edge branch consists of edge modules that aggregate local edge features and upsampled edge features, and the output edge features are also used to update point features. By helping feature extraction in the other branch, point and edge features become more powerful in the final prediction.

2.4 Scalable GCN

The previous mentioned methods have primarily focused on improving the performance of GNN on segmentation tasks. Yet few of them consider the time and memory efficiency of the model. To achieve better performance, they use expensive local feature learner like kernelization or graph construction (i.e., HPEIN), and commonly use expensive point sampling methods like Farthest Point Sampling (FPS) that is of $O(N^2)$ time complexity. Yet the common issue of such computationally expensive high-performance models is that they do not scale well to large-scale point clouds, like the outdoor scene dataset KITTI. Thus to address the scalability

issue of the previous methods, another type of networks that focuses on improving the scalability of methods is born. RandLA-Net is one of these models.

2.4.1 *RandLA-Net*

RandLA-Net[5] is a memory and computationally efficient neural architecture for large-scale point clouds. It still follows the hierarchical structure of PointNet++, and its decoding stage uses the same kNN interpolation and skip conncetions to propagate the point features. Yet compared to PointNet++, it contains much less parameters and can process 1 million points in a single pass. Unlike other hierarchical models that commonly uses farthest point sampling for downsampling, it has $O(1)$ time complexity and does not demand extra memory. Yet compared to FPS, random sampling can easily discard points of important information, especially for objects with sparse points. Thus to overcome this drawback of random sampling, RandLA-Net proposes a new dilated residual block that can effectively extract local features and significantly increase receptive fields.

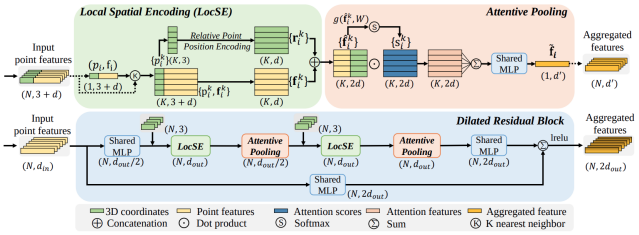

Figure 7: The architecture of Dilated Residual Block.

The proposed dilated residual block of RandLA-Net consists of two modules: local spatial encoding and attentive pooling. The local spatial encoding first finds kNN neighbors of input points and then encode the relative point position of each neighbor in the following way:

$$\mathbf{r}_i^k = MLP\left(p_i \oplus p_i^k \oplus \left(p_i - p_i^k\right) \oplus \|p_i - p_i^k\|\right)$$

Then we concatenate this relative position embedding with the point feature and get a set of point neighborhood feature $\hat{\mathbf{F}}_i$ which encodes the local geometric structures of center point pi:

$$\hat{\mathbf{F}}_i = \left\{\hat{\mathbf{f}}_i^1 \cdots \hat{\mathbf{f}}_i^k \cdots \hat{\mathbf{f}}_i^K\right\}, \quad \hat{\mathbf{f}}_i^k = \mathbf{f}_i^k \oplus \mathbf{r}_i^k$$

Then these point neighborhood features are attentively aggregated by the attention scores computed by function g. function g is basically a shared MLP followed by $softmax$. These learnt attention scores can be regarded as a soft mask which automatically selects the important features. Formally, these features are weighted summed as follows:

$$\tilde{\mathbf{f}}_i^k = \sum_{k=1}^{K} \hat{\mathbf{f}}_i^k \odot \mathbf{s}_i^k, \quad \mathbf{s}_i^k = g\left(\hat{\mathbf{f}}_i^k, \mathbf{W}\right)$$

To summarize, for the i-th point p_i in the given point cloud, the LocSE and Attentive Pooling units learn to aggregate the geometric patterns and features of its K nearest points, and finally generate an informative point feature vector $\tilde{f}_i$.

Eventually, as shown in Figure 7, the dilated residual block is constructed through stacking two sets of LocSE and Attentive Pooling units and adding a skip connection. This design is inspired by ResNet [3] and dilated networks[2]. This efficient local feature aggregation module eventually manages to capture complex local structures over progressively smaller point-sets.

In Table 2, RandLA-Net that it is possible to efficiently and effectively segment large-scale point clouds by using a lightweight network architecture. Both PointNet++[13] and PointCNN[12] are computationally expensive mainly because of the FPS sampling operation. Thanks to the simple random sampling together with the efficient MLP-based local feature aggregator, RandLA-Net, despite having similar model size as PointNet++, takes the shortest time (185 seconds) to infer the semantic labels for each large-scale point cloud (up to 10^6 points).

	Total time (seconds)	Parameters (millions)	Maximum input points (millions)
PointNet	192	0.8	0.49
PointNet++	9831	0.97	0.98
RandLA-Net	**185**	1.24	**1.03**

Table 2: Runtime comparison for semantic segmentation task on Sequence 08 of the KITTI dataset

2.5 Model Summary

PointNet is one of the earlier models to propose the possibility of consuming raw data points on the point cloud. PointNet extracts vertex features by directly applying MLP layer on each vertex in the graph.

PointNet++ observes the drawbacks of PointNet and develops a hierarchical architecture similar to FPN [11] in the 2D image segmentation task. To deal with the large amount of data in the point cloud, PointNet++ proposes to use the farthest point sampling to subsample the points from the point cloud and interpolate the points back. Many later works follow the architecture design in PointNet++.

DGCNN and LDGCNN observes the graph in the point cloud is mutually constructed, and should be reconstructed at each hidden layer to obtain a larger receptive field. The *EdgeConv* operation generalizes the PointNet operation and defines a great graph convolution operation that is widely used today.

DeepGCN further extensively studies the skip connection design and tries to enlarge the receptive field of the convolution by performing dilated k-NN.

RGCNN adopts the graph convolution operation in the spectral domain to aggregate vertex information up to k-th hop from the central vertex. It chooses a completely different graph construction scheme to tradeoff between time and space complexity.

GAC-Net follows the PointNet++ architecture and performs graph convolution by utilizing the attention mechanism. It is also partially dynamic GCNN as it performs graph reconstruction and subsampling alternatively in the network layers.

HEPIN designs a new edge branch that hierarchically encodes edge features and interacts with the point branch. HEPIN constructs the hierarchical graph when upsampling and upsampled edge features.

HDGCN adopts the depth-wise convolution from MobileNet [4] to remove the memory footprint.

RandLA-Net uses random sampling instead of farthest point sampling to achieve efficient time complexity. The dilated residual block also helps to extract different input characteristic.

3 BENCHMARK AND EVALUATION

3.1 Datasets

ShapeNet: ShapeNet is a dataset of synthetic 3D objects of 55 common categories. It was curated by collecting CAD models from online open-sourced 3D repositories. Part annotations were added to a subset of ShapeNet models segmenting them into 2-5 parts.
S3DIS: Stanford Large-Scale 3D Indoor Spaces (S3DIS) dataset comprises 3D scans of 6 large-scale indoor areas collected from 3 office buildings. The scans are represented as point clouds and annotated with semantic labels of 13 object categories.
KITTI: The KITTI benchmark is one of the most influential datasets in autonomous driving and has been commonly used in both academia and industry. It contains 7481 annotated training scenes and 7518 testing scenes without GT annotations. Frequently used train/validation split to divide the annotated part of the dataset into a train split of 3712 scenes and an validations split of 3769 scenes.
Semantics3D: Semantics3D consists of dense point clouds acquired with static terrestrial laser scanners. It contains 8 semantic classes and covers a wide range of urban outdoor scenes: churches, streets, railroad tracks, squares, villages, soccer fields and castles.

3.2 Tasks

We conduct evaluation on two tasks, part segmentation and semantic segmentation. Part segmentation studies the belonging of the points in an object. It aims at assigning labels to different parts of an object. The semantic segmentation is at the scene level. It aims at tell which object in the scene the point belongs to.

3.3 Evaluation Metrics

We evaluate the model mainly using class-wise mean of intersection over union (mIoU), considering the unbalanced distribution among classes. We also report the result of class-wise mean of accuracy (mAcc), while this metric cannot fully reflect the difference of structure.Without considering the unbalanced distribution, we also report thr point-wise overall accuracy (OA)

3.4 Results

When comparing the general structure of the models, there is a dichotomy between dynamic-graph structure that is proposed by DGCNN [16] and the hierarchical encoder-decoder architecture proposed by PointNet++[13]. It is obvious that for the scene segmentation task, the hierarchical architecture consistently performs better than the dynamic-graph structure. Although DGCNN performs better than PointNet++, it is highly likely that this improvement is because the graph convolution used in edge convolution blocks of DGCNN is more effective in extracting local geometric structures than the PointNet used by PointNet++, which is a point-wise MLP.

Method	PointNet	PointNet++	DGCNN	LDGCNN	RGCNN
mIoU	83.7	85.1	85.2	85.1	84.3

Table 3: Results on ShapeNet

The reason we think that the hierarchical encoder-decoder structure performs significantly better than dynamic-graph structure on scene segmentation tasks is that, the point downsampling process in the hierarchical structure significantly increase the receptive field size and thus can encode geometric structure better in large point cloud. In contrast, dynamic graph updates never downsamples points and the point size remains the same across the whole structure. Therefore, its receptive field increases much slower than dilation and can be rather ineffective in large.

Method	S3DIS (6-fold)			S3DIS (area-5)		
	OA	mAcc	mIoU	OA	mAcc	mIoU
PointNet	78.5	66.2	47.6	-	48.98	41.09
PointNet++	81.0	67.1	54.5	-	-	-
DGCNN	84.1	-	56.1	-	-	-
DeepGCN (ResGCN-28)	85.9	-	60.0	-	-	52.49
GACNet	-	-	-	87.79	-	62.85
HPEIN	88.2	76.26	67.83	87.18	68.3	61.85
HDGCN	-	76.11	66.85	-	65.81	59.33
RandLA-Net	88.0	82.0	70.0	-	-	-

Table 4: Results on S3DIS

Another phenomenon we notice is that, the two best performing models are GACNet and RandLA-Net, which all uses attentive aggregation in their local feature learner. This showcases that compared to max-pooling or summation, attention-based aggregation can better encode local geometric structure of point clouds.

Method	Semantic3D (reduced-8 challenge)	
	OA	mIoU
GACNet	91.9	70.8
RandLA-Net	94.8	77.4

Table 5: Results on Semantic3D (reduced-8 challenge)

When we are comparing ResGCN and DGCNN, their results on S3DIS showcases that ResGCN achieved a significant performance improvement. And when we compare results of GACNet and RandLA-Net on Semantic3D, it seems that RandLA-Net, which has a very similar attention-based local feature learner as that of GACNet, also performs better. The common difference between these two pairs is that, the model that performs better both added skip connections to their stacked feature abstraction block. Thus in regard to the structure of feature extraction block, adding skip

Method	SemanticKITTI	
	params (M)	mIoU
PointNet	3	14.6
PointNet++	6	20.1
RandLA-Net	1.24	53.9

Table 6: Results on SemanticKITTI

connections is effective in helping hierarchical network learn the geometric structures in the point cloud.

In terms of model scalability and speed, the results of RandLA-Net in Table 2 demonstrated that the sampling cost is a huge bottleneck for the previous hierarchical encoder-decoder structure models, and it is possible to substitute expensive Farthest point sampling (FPS) with random sampling can significantly shorten runtime while keeping similar if not better performance. although it's less likely for FPS to miss important points in the point cloud than random sampling, this drawback can be overcome with more effective local feature encoder.

4 FRONTIER AND VISION

We can tell from the result comparison that when it comes to performance improvement on segmentation tasks, the point sampling and grouping methods choices barely influence the final performance, and most performance boosts come from improvement on local feature learner. Thus to further improve the performance of GNN on larger point clouds, we'd better concentrate on improving the local feature encoding block instead of resource expensive data dependent point sampling and graph reconstruction between layers. Due to the recent success of transformer architecture on both natural language tasks and vision tasks, a direction would be using transformer to encoder local spatial structure of a point neighborhood.

As for the long-term vision of GNN on point cloud segmentation tasks, considering that recently most 3D point cloud data come from LiDAR scanners, and they are widely applied in autonomous driving where real-time segmentation is necessary. How to further improve the models' precision on large-scale outdoor point clouds, and how to lessen the computation cost to achieve real-time segmentation will remain as two significant challenges for point cloud segmentation methods.

5 CONCLUSION

In conclusion, this survey discusses a series of graph convolution neural network models on point cloud segmentation. Point-based methods starts the network architecture that directly consumes point cloud raw data. Dynamic Graph based methods instead focuses on developing effective graph convolution operation and perform dynamic graph reconstruction at each layer. Hierarchical-based methods adopts the PointNet++ architecture and utilizes graph convolution operations to aggregate information from neighboring vertices. Scalable GCN methods seeks for a efficient and effective graph convolution operation and subsampling methods that can be applied on large point cloud. Given all the above GCN methods, there are two significant factors that improve the performance of the model. The first factor is the local feature aggregator. The attention mechanism is an effective aggregator compared to the simple point-wise MLP in PointNet. The second factor is the encoder-decoder architecture or the feature pyramid network architecture. The architecture effectively models the input characteristic at multi-scale to adopt objects of different sizes in the point cloud. The subsampling at each layer also significantly helps the model to aggregate more information locally, or in other words, obtain a larger receptive field.

REFERENCES

[1] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf

[2] Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. 2020. Dilated point convolutions: On the receptive field size of point convolutions on 3D point clouds. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 9463–9469.

[3] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

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

[5] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. 2020. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11108–11117.

[6] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2261–2269. https://doi.org/10.1109/CVPR.2017.243

[7] Li Jiang, Hengshuang Zhao, Shu Liu, Xiaoyong Shen, Chi-Wing Fu, and Jiaya Jia. 2019. Hierarchical Point-Edge Interaction Network for Point Cloud Semantic Segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 10433–10441.

[8] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations* (Palais des Congrès Neptune, Toulon, France) *(ICLR '17)*. https://openreview.net/forum?id=SJU4ayYgl

[9] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. DeepGCNs: Can GCNs Go As Deep As CNNs?. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[10] Z. Liang, M. Yang, L. Deng, C. Wang, and B. Wang. 2019. Hierarchical Depthwise Graph Convolutional Neural Network for 3D Semantic Segmentation of Point Clouds. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 8152–8158. https://doi.org/10.1109/ICRA.2019.8794052

[11] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. 2017. Feature Pyramid Networks for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 936–944. https://doi.org/10.1109/CVPR.2017.106

[12] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 652–660.

[13] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf

[14] Gusi Te, Wei Hu, Amin Zheng, and Zongming Guo. 2018. RGCNN: Regularized Graph CNN for Point Cloud Segmentation. In *Proceedings of the 26th ACM International Conference on Multimedia* (Seoul, Republic of Korea) *(MM '18)*. Association for Computing Machinery, New York, NY, USA, 746–754. https://doi.org/10.1145/3240508.3240621

[15] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. 2019. Graph Attention Convolution for Point Cloud Semantic Segmentation. In *Proceedings*

of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 10296–10305.

[16] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. Graph.* 38, 5, Article 146 (Oct. 2019), 12 pages. https://doi.org/10.1145/3326362

[17] Kuangen Zhang, Ming Hao, Jing Wang, Clarence W. de Silva, and Chenglong Fu. 2019. Linked Dynamic Graph CNN: Learning on Point Cloud via Linking Hierarchical Features. arXiv:1904.10014 [cs.CV]