Graph Convolutions Using Local Structures in Feature Space for Graph Neural Networks

TOKYO UNIVERSITY OF AGRICULTURE AND TECHNOLOGY
Department of Electronic and Information Engineering

Yang Li

Supervisor: Prof. Takahumi Saito

February 3, 2023

Abstract

In this dissertation, graph convolutions using local structures in feature space for graph neural networks are studied.

How to process irregular data is one of the hot topics in signal processing. Most irregular data can be represented using graph data.Therefore, graph signal processing is also a key research topic in signal processing. Many graph signal processing methods have been proposed.

On the other hand, deep neural networks have achieved remarkable success in processing regular signals, such as images and videos. Therefore, using deep neural networks to process graph data has received much attention. Because classical deep neural networks can not process graph data directly, Graph neural networks (GNNs) are proposed. The key to graph neural networks is to design the graph convolution methods (GCs), i.e., graph filters. Moreover, the expressiveness of GCs is that if we can recognize the nodes by their node-wise features.

Existing GCs may occur over-smoothing. Over-smoothing has a tendency to make the node-wise characteristics of nodes the same. Therefore, over-smoothing reduces the expressiveness of GCs. Furthermore, it also influences the performance of GNNs. To reduce the possibility of occurring over-smoothing, a resolution is to improve the expressiveness of GCs. Existing GCs still have some expressive limitations:

1. Existing methods primarily focus on using node-wise features of one-hop neighborhood and ignore the *structural information* of the surrounding neighboring nodes, particularly in the feature space.

2. While the multi-hop neighborhood may contain some useful information, this cannot be utilized for the existing single step GC.

3. For attention-based spatial graph convolutions, they primarily use one type of attention function. However, different types of attention functions get different attention weights. These may influence the expressive of methods. Existing attention spatial methods do not consider this influence.

In other words, if we can breakthrough these limitations, the expressiveness of GCs can be improved.

The goal of this dissertation is to propose some high-expressive spatial GCs for GNNs. In details, we focus on using more detailed local information in a GC than existing GCs. Therefore, we first proposed some structural features which describe the structural information in feature space. Then we construct three GCs that have high-expressive by using local structures in feature space.

This dissertation is organized as follows. Chapter 1 introduces the motivation and the limitations of expressive of the existing GCs. The basis knowledges and related works are described in Chapter 2. The main contributions of this dissertation are described in Chapter 3, Chapter 4 and Chapter 5, whose summaries are represented as follows.

In Chapter 3, we focus on the limitation 1. We first proposed three structural features—feature angle, feature distance, and relational embedding—to describe the structural

information of the surrounding neighboring nodes in the feature space. Then, we introduce these structural features into spatial GC to improve the expressiveness of GC. We call this new GC—Structure-aware Graph Convolution, i.e., SAGConv.

In Chapter 4, we focus on the limitation 2. We propose a spatial GC aggregating multi-hop features. In contrast to the iteration of one-hop feature aggregation, multi-hop features are aggregated simultaneously in the one-step GC in our method. We utilize this spatial GC and the structural features simultaneously to present our Structure-aware Multi-hop Graph Convolution (SAMGC).

In Chapter 5, we focus on the limitation 3. We propose an attention function utilizing multi-type attention functions simultaneously. Therefore, it can get better attention weights. We simultaneously use the proposed attention function and structural features to present our attention-based spatial GC—Multi-type Structure and Position-aware attention-based Graph Convolution (MTSPAGC).

Finally, this dissertation is concluded in Chapter 6.

Contents

Chapter 1

Introduction

Information technology (IT) has advanced quickly so far, with major IT companies like Apple, Microsoft, Facebook, and Google driving this development. These days, social networking services have been released, with Facebook, Twitter, and other prominent new companies. We can now do a lot using electronic (mobile) devices thanks to the above developments. For instance, someone can earn money by uploading his or her videos to video-sharing services like YouTube. The top YouTube creators, sometimes known as YouTubers, can earn over one million dollars each year [1]. Due to the coronavirus epidemic, now, many people are working online. These two instances suggest that work styles are fast changing as a result of IT developments.

Signal processing is a fundamental part of such recent developments of IT-related technologies [2–13]. Using deep neural networks to process signals is a hot topic in signal processing. For example, in automatic driving and robotic vision, we use a digital camera to get digital images of scenes, i.e., image signals, then use deep neural networks to recognize roads or find target objects. Due to the excellent performance and the developments of computing techniques, deep neural networks have been utilized in many fields [14]. However, as shown in Fig. 1.1, the traditional deep neural networks can only process regular data, such as pictures and videos [15–17]. There are many irregular data in the real world, such as societal relationship of people, 3D point clouds, the number of passengers in traffic networks. In order to utilize deep neural networks to process irregular data, the earliest efforts were to replace irregular data with regular data approximations. However, the approximations may lose some information and increase the complexity. To process irregular data efficiently, they are always represented using graph data. Graph data is a kind of data structure that models a set of objects (nodes) and their relationships (edges). The irregular data can be represented using graph data easily, naturally, and straightforwardly. Graph data is also an efficient data structure. We can add or remove nodes or edges at any time. Therefore, graph data are wildly used in many fields, such as friends recommending social networking services, for example, Facebook or Twitter, and goods recommending services in online shopping, such as Amazon. On the other hand, inspired by the performance of traditional deep neural networks to process regular data, graph neural networks (GNNs) are also proposed by researchers for efficiently processing graph data. Since 2013, a lot of GNNs have been proposed, such as spectral network [15], ChebNet [18], GCN [19], and so on. GNNs are also be applied in many tasks, such as recommender systems [20], action recognition [21], traffic forecasting [22], molecular optimization [23], and so on. Recently, some researchers also try to utilize GNNs to process regular data like images [24]. The most important part of a GNN is the graph convolution operation (GC), i.e., the graph filter. Furthermore, the expressiveness of GCs is that: If we can recognize the nodes using their node-wise features. How to improve the expressiveness of GCs is still an open challenge.

Therefore, in this dissertation, we also focus on the problem—how to improve the ex-

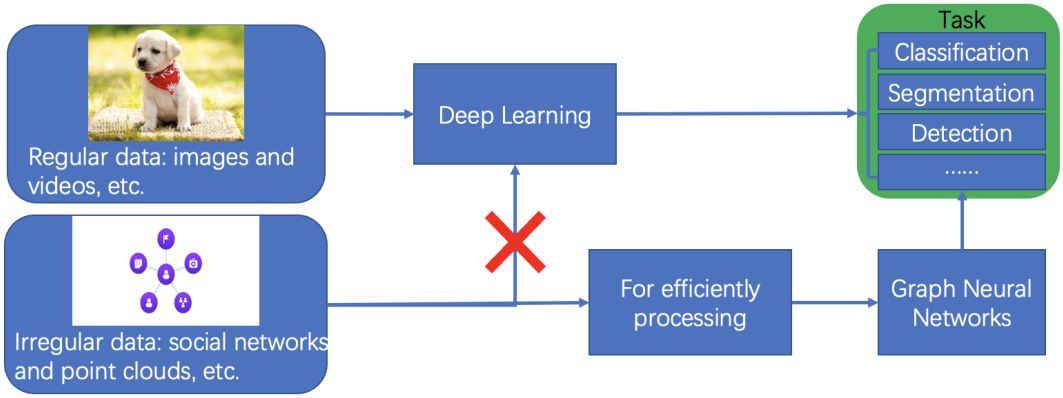

Fig. 1.1. In the real world, there are many irregular data, these irregular data can not be processed by traditional deep neural networks but may be processed efficiently by transforming to graph data and using graph neural networks.

pressiveness of GCs for GNNs.

1.1 Graph neural networks and limitations

GNNs consist of GC layers. In other words, GCs, also known as graph filters, are the base part of GNNs. Therefore, to build a high-performance graph neural network, the key is to construct a high-expressive GC. In the priors, the GCs can be classified into spectral and spatial methods. Furthermore, the spatial methods can be divided into normal spatial methods and attention-based spatial methods. However, the primary concept behind GC algorithms is to iteratively aggregate node-wise features from neighbors before integrating the aggregated information with that of the target node [25–27]. This mechanism is also called message passing mechanism. According to this mechanism, GCs have two parts, i.e. aggregator and integrator. The aggregator first collects information of neighboring nodes, then it aggregates collected information. The integrator update the node-wise features of a target node using the output of aggregator and the node-wise features of the target node.

Existing methods occur over-smoothing [25, 28]. Here, we use an example, which shows in Fig.1.2, to introduce the over-smoothing. In this example, a spatial GC is utilized to process the signals (node-wise features) of node $v1$ and the signals (node-wise features) of node $v2$. This GC uses a mean aggregator. We can see that the output of aggregator on node $v1$ and the output of aggregator on node $v2$ are the same. Therefore, the updated node-wise features of node $v1$ and the updated node-wise features of node $v2$ also trends to the same. After several rounds of processing, the node-wise features of node $v1$ and the node-wise features of node $v2$ are the same. Therefore, we can not distinguish node $v1$ and node $v2$ using node-wise features. This situation is called over-smoothing. Over-smoothing reduces the expressiveness of GCs. Furthermore, it influences the performance of GNNs.

Therefore, to improve the performance of GNNs, we should improve the expressiveness of GCs. We think existing methods have some expressive limitations as follows:

1. Existing methods primarily focus on using node-wise features of one-hop neighborhood, such as GCN [19], GraphSAGE [29], GAT [30] and so on. Therefore existing methods might lose the *structural information* of the surrounding neighboring nodes, particularly in the feature space.

2. While the multi-hop neighborhood may contain some useful information, this cannot be utilized for the single step graph convolution.

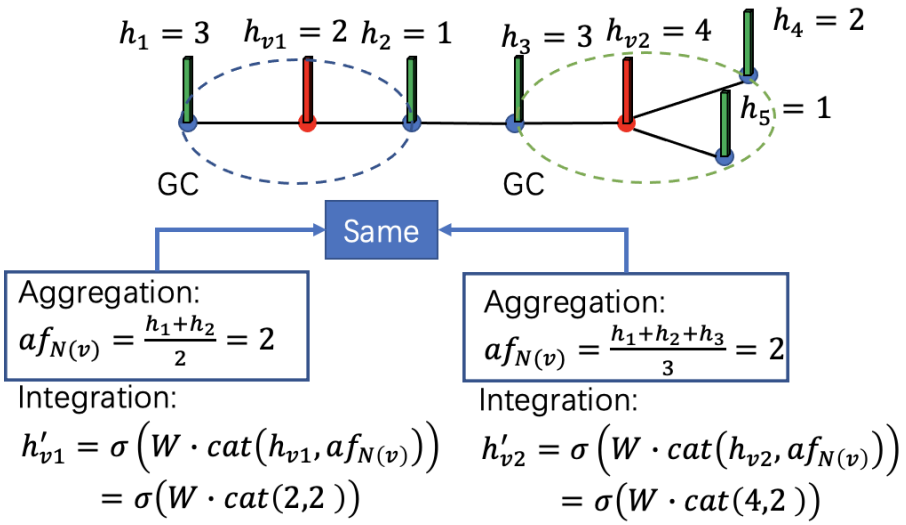

Fig. 1.2. The example of over-smoothing. In this example, a spatial GC is used. This GC uses a mean aggregator. We do GC in node $v1$ and node $v2$. The outputs of aggregator are the same. Therefore, the updated node-wise features of node $v1$ and the updated node-wise features of node $v2$ also trend to the same. This situation is called over-smoothing.

3. For attention-based spatial methods, they primarily use one type of attention function, i.e., dot-production, subtraction, or concatenation. However, different types of attention functions get different attention weights. These may influence the performance of the method. Existing attention-based spatial methods do not consider this influence.

The first limitation means that if we can efficiently use the detailed structural information of the surrounding neighboring nodes the expressive of GCs may be improved.

The second limitation also means that the useful information contained in the multi-hop neighborhood may be helpful for improving the expressive of GCs.

The third limitation means that if we utilize multi-type attention functions when we calculate the attention weights for attention-based spatial methods, it may get better attention weights. Furthermore, the expressive of attention-based spatial methods may be improved.

1.2 Objective of this study

The objective of this dissertation is to propose some high-expressive GCs for GNNs. In detail, we focus on using more detailed local information in a GC than existing GCs. Therefore, we first proposed some structural features which describe the structural information in feature space. Then we construct three GCs that have high-expressive by using local structures in feature space.

In existing GCs, they can be divided into spectral methods and spatial methods. For spectral GCs, in the domain of the graph Fourier, a GC is carried out. The graph Fourier basis is the eigenvector matrix of the graph Laplacian [15, 31]. Polynomial approximations are usually employed instead of eigendecomposition because eigendecomposition requires high computing complexity for big graphs [18, 32, 33]. It should be noted that most spectral graph convolutions rely on a fixed graph for both training and testing. Because of this, spatial graph convolution may fit time-varying signals better and can be applied to more tasks. Therefore, we also focus on spatial methods. Spatial methods can further be divided into simple spatial methods and attention-based spatial methods. As we shown in Fig. 1.3, in this study, two simple spatial graph convolution methods are actually studied. Furthermore,

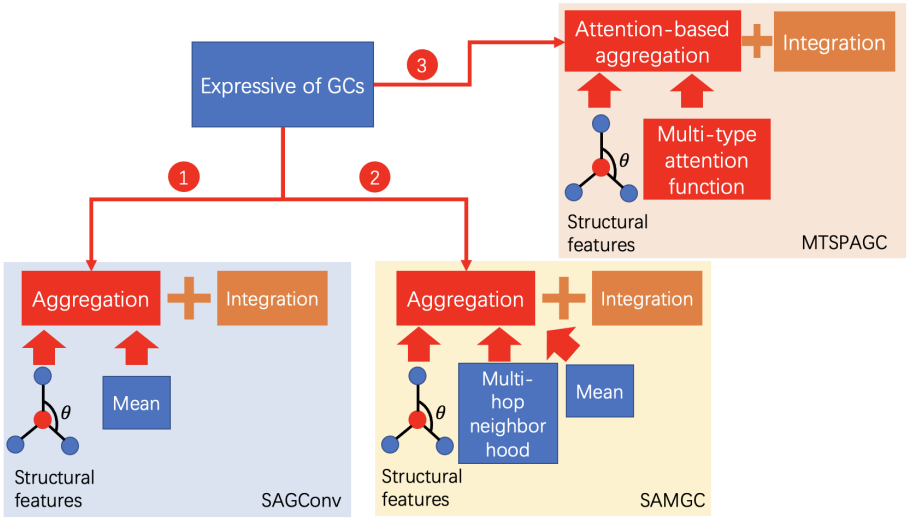

Fig. 1.3. In this study, two simple spatial graph convolution methods and one attention-based spatial graph convolution method are actually studied. They both use local structures in feature space. They also consider limitations which are described before.

an attention-based spatial graph convolution method also is proposed. These abstracts are indicated as follows:

1 **Structural features and spatial graph convolution (Chapter 3 and [34])**

To improve the expressive of normal spatial GCs and classify signals on graphs, we suggest structural features for spatial GC [34].

GCs, also known as graph filters, are frequently utilized in existing GNNs. The fundamental principle of graph convolution techniques is to integrate the aggregated features with that of the target node by first iteratively aggregating features from neighbors [25–27]. However, current approaches mostly concentrate on utilizing node-wise features [19,29,30]. This could occur over-smoothing, furthermore influence the performance of GNNs. In other words, the performance of GNNs may be further enhanced if we can reduce the possibility of over-smoothing by improving the expressive of GCs.

To improve the expressive of GCs, we provide three structural features for describing the structure of the surrounding neighboring nodes of the target node in feature space. We then propose a new spatial GC algorithm called structure-aware graph convolution by integrating the structural information into spatial graph convolution (SAGConv).

2 **Structural features and multi-hop spatial graph convolution (Chapter 4 and [35])**

To furthermore improve the expressive of normal spatial GCs and classify signals on graphs, we suggest structure-aware multi-hop spatial graph convolution [35].

As we described before, the primary idea of graph convolution algorithms is to integrate the aggregated features with that of the target node by iteratively aggregating features from neighbors [25–27]. In the single step of graph convolution, existing methods mostly concentrate on utilizing the node-wise features of one-hop neighborhood [19, 29, 30]. Therefore, there are two limitations of expressive: 1) Existing methods might lose the structural information of the surrounding neighboring nodes, particularly in the feature space. 2) While the multi-hop neighborhood may contain some useful information, this cannot be utilized for the single step GC. In other words, if we can effectively use 1) comprehensive structure information of the nearby neighboring

nodes in the feature space and 2) graph convolutions that take into account multi-hop neighborhoods, the expressive of GCs may be further improved. Therefore, the performance of GNNs may also be further improved.

To improve the expressive of GC, our proposed GC has two methods. First, we utilize three structural features, i.e., feature angle, feature distance and relational embedding, to describe the structure in the feature space. We concatenate them with node-wise features. Second, we propose a graph convolution aggregating multi-hop features. In additional, to fully utilize one-hop neighborhood information during aggregation, we also propose a neighbor-wise learnable average aggregation. We simultaneously utilize these methods to present our structure-aware multi-hop graph convolution (SAMGC).

3 **Multi-type structure and position-aware attention-based graph convolution (Chapter 5)**

To propose a high-expressive attention-based spatial GC and classify signals on graph, we also suggest multi-type structure and position-aware attention-based graph convolution.

Spatial GC methods can be divided into normal spatial GCs and attention-based spatial GCs. The core idea of the normal and attention-based spatial GCs are the same, i.e., integrate the aggregated features with that of the target node by iteratively aggregating features from neighbors [25–27]. But in the attention-based spatial graph convolutions, the attention weights for neighboring nodes of a target node is introduced. Therefore, attention-based spatial GCs can more efficiently utilize the information of neighboring nodes by treating neighboring nodes differently. However, existing attention-based spatial graph convolutions also focus on using the node-wise features of neighborhood and only utilizing one type of the attention function when calculating attention weights [19, 36–38]. Therefore, there exist two limitations of expressive: 1) Existing methods might lose the structural information of the surrounding neighboring nodes, particularly in the feature space. 2) Different types of attention functions may get different attention weights, these weights may affect the expressive of the method. In other words, if we can effectively use 1) comprehensive structure information of the nearby neighboring nodes in the feature space and 2) multi-type attention functions simultaneously, the expressive of attention-based GCs may be further improved.

To achieve these goal, our attention-based graph convolution has two methods. In the first method, three structural features—feature angle, feature distance, and relational embedding—are defined in the feature space. The second method utilizes multiple attention functions simultaneously in one attention-based graph convolution. Both approaches can be utilized simultaneously to present our multi-type structure and position-aware attention-based graph convolution (MTSPAGC).

This dissertation is organized as follows. The general preliminaries related to our study are introduced in Chapter 2. This Chapter introduces some basic concepts and knowledges and existing methods. The main parts of this dissertation are described in Chapter 3, Chapter 4, and Chapter 5. Finally, we conclude this study in Chapter 6.

Chapter 2

Basic knowledge

In this section, we introduce the basic knowledge of our method and some existing methods. We first introduce the concept of graph data. Second, we introduce the characters of the graph data. Third, we describe the graph signal processing. Then, we introduce classic neural networks. Finally, we introduce the graph neural networks (GNNs).

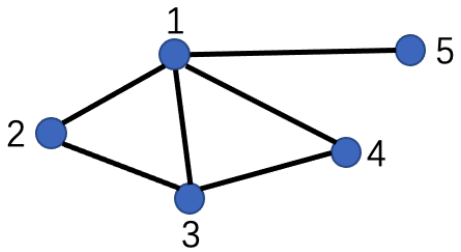

Fig. 2.1. The example of graph data, the set of vertices is $V = \{1, 2, 3, 4, 5\}$, the set of edges is $E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{3, 4\}\}$.

2.1 Concept of graph data

The graph data can be presented as $G = (V, E)$, V is vertices or nodes or points, E is the set of edges or lines. In Fig.2.1, we drew a graph.

Typically, a graph is displayed by placing a dot at each vertex and, if the matching two dots have an edge, drawing a line to connect them. The vertex of the set of vertices is presented as $v \in V$, the edge of the set of edges is presented as $e \in E$. The vertex set of a graph G is referred to as $V(G)$, the edge set of a graph G is referred to as $E(G)$. Finally, all the edges connected to vertex v is presented as $E(v)$. If vertices x, y of G have an edge, they are adjacent, or neighbors. If all the vertices of G are pairwise adjacent, then G is a complete graph. Pairwise non-adjacent vertices are called independent.

If the types of vertices are same, graph G is a homogeneous graph. In Fig.2.2, we show an example of homogeneous graph. In this example, each vertex represents a paper, each edge represents the citation relationships.

If the graph G can compose different types of vertices, the graph G is a heterogeneous graph. We show an example of heterogeneous graph in Fig.2.3. In this heterogeneous graph, two vertices represent authors of a paper, three vertices represent papers. The edges represent writing relationships.

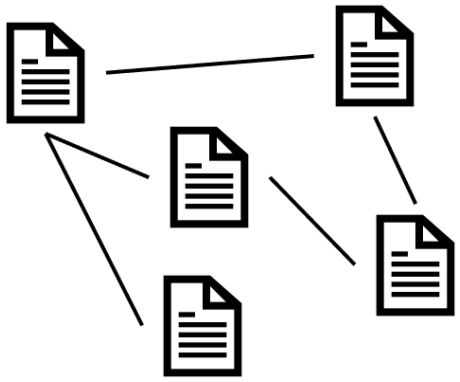

Fig. 2.2. The example of a homogeneous graph, citation graph. Each vertex represents a paper. Each edge represents the citation relationships.

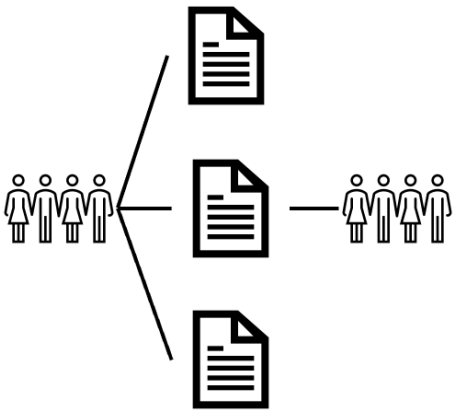

Fig. 2.3. The example of a heterogeneous graph. Two vertices represent authors of a paper, three vertices represent papers. The edges represent writing relationships.

2.2 The characters of graph data

In the graph $G = (V, E)$, the number of vertices $\mid V \mid$ is the order of graph G, where $\mid \cdot \mid$ represents the number of components in a set. According to the order of graph G, graph can be regarded as finite, infinite and countable. In our research, graphs are finite.

The set of neighbors of a vertex v in the graph G is represented as $N_G(v)$. Here, for briefly, we just use $N(v)$. The degree of vertex v, $d_G(v) = d(v)$, is the number of edges connect with v. Here, according to definition of a graph, the degree of vertex v, $d(v)$, is equal to the number of neighbors of v, as follows:

$$d(v) = N(v) \tag{2.1}$$

If the degree of vertex v is 0, i.e.$d(v) = 0$, we call the vertex v is an isolated vertex.

The number

$$\delta(G) := min\{d(v) \mid v \in V\}, \tag{2.2}$$

Is the minimum degree of G, the number

$$\Delta(G) := max\{d(v) \mid v \in V\}, \tag{2.3}$$

Is the maximum degree of G. The number

$$d(G) := \frac{1}{\mid V \mid} \sum_{v \in V} d(v). \tag{2.4}$$

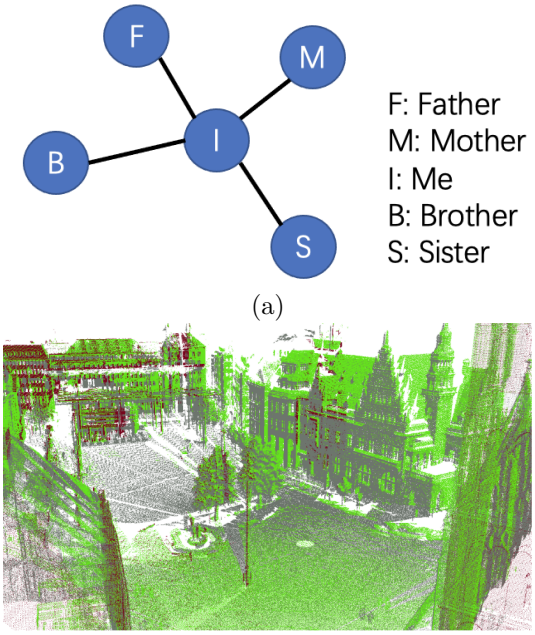

Fig. 2.4. The examples of irregular data, (a) is the social networks, (b) is the 3D point clouds [39].

is the average degree of G. Clearly,

$$\delta(G) \leq d(G) \leq \Delta(G). \tag{2.5}$$

If all the vertices of G have the same degree k, then G is k-regular.

Adjacency matrix W is utilized to represents the adjacency relationships of vertices. The adjacency matrix W is defined as follows:

$$W := [w_{ij}]_{|V|\times|V|}, \tag{2.6}$$

where w_{ij} is the weight on the edge between vertex i and vertex j. If the graph G is an unweight graph, the value of w_{ij} is defined as follows:

$$w_{ij} = \begin{cases} 1, & \text{if vertex } i \text{ is connecting with vertex } j, \\ 0, & \text{if vertex } i \text{ is not connecting with vertex } j. \end{cases} \tag{2.7}$$

Due to the relationship between vertices, graph also can be divided into directed graph or undirected graph. The direction usually represents the direction in which the message is delivered.

2.3 Graph signal processing

As we all know, classical signal processing (CSP) is wildly utilized in processing regular data, such as image and audio signals. Regular data are set to regular grade. Recently, how to process irregular data, such as 3D point clouds, sensor networks, brain networks, social networks, and so on, has received much attention. In counter to regular data, irregular data do not have regular grades. The examples of irregular data are shown in Fig.2.4. Therefore, irregular data are difficult to be processed using CSP. When processing irregular data, they may be processed efficiently by converting them into graph-structured data (graph signals).

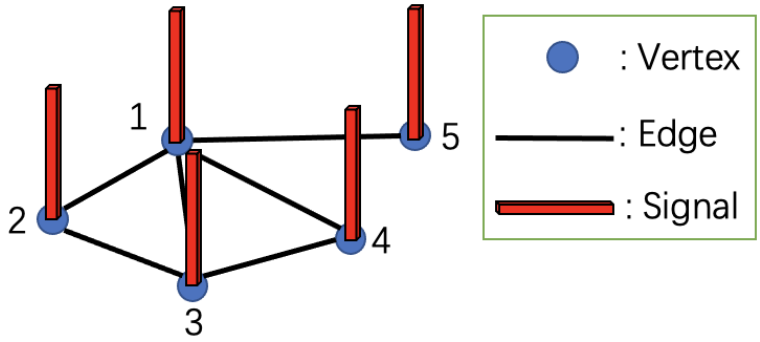

Fig. 2.5. Example of graph signals.

In Fig.2.5, we show an example of graph signals. To process graph signals, GSP [26, 40–42] has been developed. GSP can process graph signals with spectral graph theory.

In GSP, signal values are a set of vertices in a graph. One vertex connects with others using edges (relationships). As described in 2.2, the graph can be a weighted graph or an unweighted graph. In a weighted graph, the edge weights usually represent the similarities between signal values. In GSP, like CSP, signals are processed in the frequency domain. However, graph signals are transformed to the frequency domain using a graph Fourier transform. Below, we review the basic theory of GSP.

2.3.1 Graph Laplacian Matrix

We consider a graph $G(V, E, W)$ is an undirected graph, where V is the set of vertices, E is the set of edges, and W is the adjacency matrix. Additionally, we also assume graph G without self-loops and it is a homogeneous graph. Due to the graph G is an undirected graph, the element $w_{ij} = w_{ji}$, if the edge between vertex i and vertex j exists. We let $D := \mathrm{diag}(d_1, ..., d_i, d_j, ...d_{|V|})$ be the diagonal degree matrix, where $d_i = \sum_{j=1}^{|V|} w_{ij}$.

The graph Laplacian matrix L is defined as follows

$$L := D - W, \tag{2.8}$$

The eigendecomposition of L is expressed as $L := U\Lambda U^{\mathrm{T}}$, where $\cdot^{\mathrm{T}}$ is the transform operation, $U = [\mathbf{u}_{\lambda_1}, \mathbf{u}_{\lambda_2}, ..., \mathbf{u}_{\lambda_i}, ..., \mathbf{u}_{\lambda_{|V|}}]$ is the matrix composed of eigenvectors $\mathbf{u}_{\lambda_i}$ and $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, ..., \lambda_i, ..., \lambda_{|V|})$ is the diagonal matrix with the corresponding eigenvalues. For simplicity, $0 = \lambda_1 < \lambda_2 < ... < \lambda_i < ... < \lambda_{|V|} = \lambda_{max}$, where λ_{max} is the maximum eigenvalue of Λ.

2.3.2 Graph Fourier Transform

In CSP, Fourier basis for the continuous signals is the $e^{j\omega t_{\mathrm{F}}}$, where ω is a normalized angular frequency and t_{F} is arbitrary time. When an arbitrary continuous signal $f(t_{\mathrm{F}})$, the classical Fourier transform is represented as follows,

$$\widetilde{f}(\omega) := \int_{\mathbb{R}} f(t_{\mathrm{F}}) e^{-j\omega t_{\mathrm{F}}} dt_{\mathrm{F}}, \tag{2.9}$$

where $\widetilde{f}(t_{\mathrm{F}})$ is the Fourier coefficient. Due to CSP, the Fourier basis $e^{j\omega t_{\mathrm{F}}}$ is the eigenfunctions of the Laplace operator. It is established as follows,

$$-\frac{\partial^2 (e^{j\omega t_{\mathrm{F}}})}{\partial t_{\mathrm{F}}^2} = \omega^2 e^{j\omega t_{\mathrm{F}}}, \tag{2.10}$$

where ω^2 is the eigenvalues of $e^{j\omega t_F}$. Due to Equation (2.10), we can know that if the frequency (eigenvalue) is low, the associated eigenfunction is slowly oscillating function. In contrast to this, if the frequency (eigenvalue) is high, the associated eigenfunction is rapidly oscillating function.

in GSP, the similar notion about the frequency is also provided. The eigenvector of L in Equation (2.8) is composed as follows,

$$\mathbf{u}_{\lambda_i} = [u_{v_1}^{\lambda_i}, ..., u_{v_j}^{\lambda_i}, ..., u_{v_{|V|}}^{\lambda_i}]^{\mathrm{T}}. \tag{2.11}$$

The values of the eigenvector are placed on the corresponding vertices, i.e. the value $u_{v_j}^{\lambda_i}$ is placed on the vertex v_j. If there is an edge between the vertices v_j and v_k and λ_i is small, the value of $u_{v_j}^{\lambda_i}$ is similar to the value of $u_{v_k}^{\lambda_i}$. Otherwise, if λ_i is large, the value of $u_{v_j}^{\lambda_i}$ is different from the value of $u_{v_k}^{\lambda_i}$. Because the eigenvector is the slowly oscillating vector if the corresponding eigenvalue is small, otherwise, the eigenvector is considered as the rapidly oscillating vector. Therefore, the relationship between the eigenvector and the eigenvalues is similar to the notion of the frequency. In GSP, the eigenvectors of the graph Laplacian matrix are considered as the Fourier basis, and the corresponding eigenvalues are considered as the frequency. The graph Fourier transform (GFT) can be defined using the eigenvectors and eigenvalues of the graph Laplacian matrix in GSP.

The GFT of the input $\mathbf{x} \in \mathbb{R}^{|V|}$ is defined as

$$\widetilde{\mathbf{x}} := U^{\mathrm{T}}\mathbf{x}, \tag{2.12}$$

where $\widetilde{\mathbf{x}}$ is the signal on the graph spectral domain. Therefore, the inverse GFT is given by $\mathbf{x} = U\widetilde{\mathbf{x}}$. Similar to CSP, we can define the graph spectral filtering as follows:

$$\widetilde{x}_i^{'} := h(\lambda_i)\widetilde{x}_i, \tag{2.13}$$

where $\widetilde{x}_i$ is the ith component of $\widetilde{\mathbf{x}}$, and $h(\lambda_i)$ is the spectral response for the ith eigenvalue. Additionally, $\widetilde{x}_i^{'}$ is the coefficient filtered by $h(\lambda_i)$ on the graph spectral domain. A graph spectral filtering with the matrix using the GFT and the inverse GFT can be given as follows,

$$\begin{aligned} \hat{\mathbf{x}} &= Uh(\mathbf{\Lambda})\mathbf{U}^{\mathbf{T}}\mathbf{x} \\ &= h(\mathcal{L})\mathbf{x}, \end{aligned} \tag{2.14}$$

where $\hat{\mathbf{x}}$ is the filtered graph signal in the spectral domain, $h(\Lambda)$ is the spectral response of a graph filter:

$$h(\mathbf{\Lambda}) := \begin{bmatrix} h(\lambda_1) & & \mathbf{O} \\ & \ddots & \\ \mathbf{O} & & h(\lambda_{|V|}) \end{bmatrix}. \tag{2.15}$$

2.4 Classic Neural Networks

To our best knowledge, the smartest animal on the earth is the human. The human brain is the best organ in the human body. Therefore, how to make a machine have a "brain" is a goal of artificial intelligence. The base part of the brain is the neuron. We show a neuron in Fig. 2.6. The mechanism of a neuron is that one neuron receives a message from other neurons, then processes this message, and finally sends the message to other neurons. In math, this mechanism can be represented as follows:

$$\mathbf{y} = f(\mathbf{x}), \tag{2.16}$$

where $\mathbf{x}$ is the input message from other neurons, $f(\cdot)$ is the process in this neuron, $\mathbf{y}$ is processed message. In Fig.2.7, we show this mechanism. One neuron connects with other

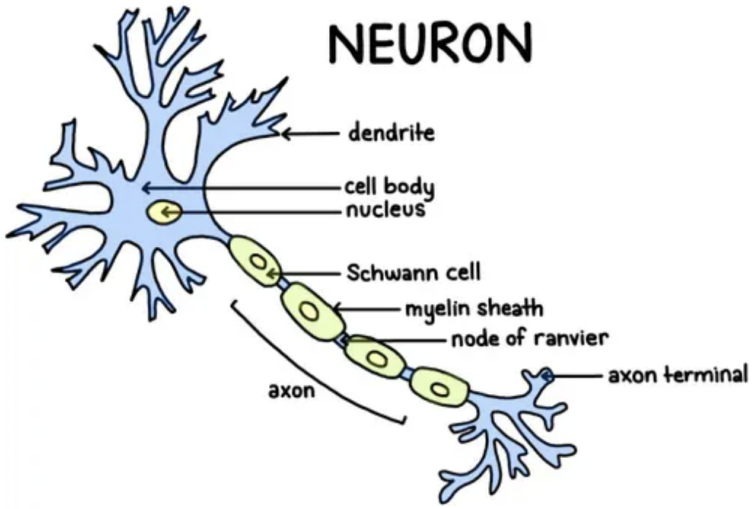

Fig. 2.6. The structure of neuron. [43]

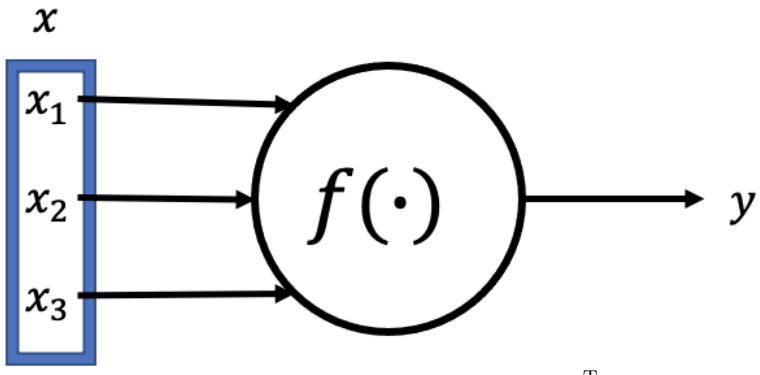

Fig. 2.7. The mechanism of artificial neuron. $\mathbf{x} = [x_1, x_2, x_3]^{\mathrm{T}}$ is the input message from other neurons, $f(\cdot)$ is the process in this neuron, $\mathbf{y}$ is processed message.

neurons, making a network. The neural network is a model that mimics the mechanism of human neurons and the construction of human neural networks. The structure of neural networks is shown in Fig. 2.8. In the structure of neural networks, a part of neurons makes a layer.

In Fig.2.8, we show the simplest neural network. In this neural network, a neuron in Layer 1 connects with neurons in Layer 2. If the scale of the network becomes large, it may need a huge computing source. Due to the fully connected structure, a neuron can not get the local features of the input. On the other hand, a neuron of the eyes has a receptive field. The receptive field of a neuron is the specific area of the retina where only stimuli within this area can activate the neuron. Inspired by this mechanism, convolutional neural networks are proposed and become the most widely utilized classic neural networks. The structure of convolutional neural networks is shown in Fig.2.9. In the follows, we first introduce the convolution layer, then we introduce the Activation layer, finally we introduce the Pool layer.

2.4.1 Convolution layer

In convolutional neural networks, a kernel convolutes with the input. In the follows, we introduce the convolution operation.

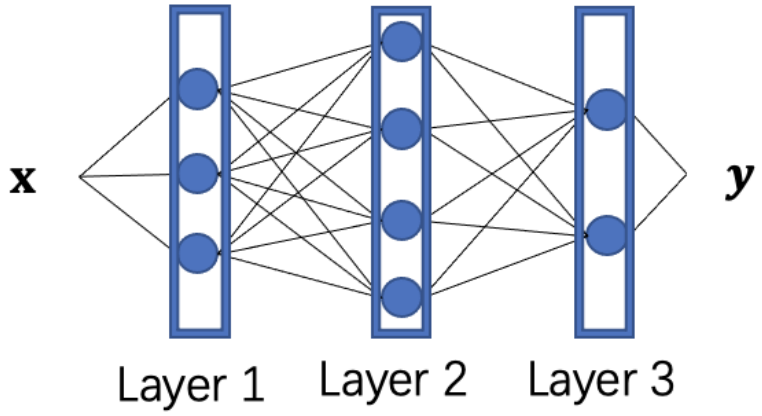

Fig. 2.8. The structure of neural networks. **x** is the input, **y** is the output. A part of neurons making a layer.

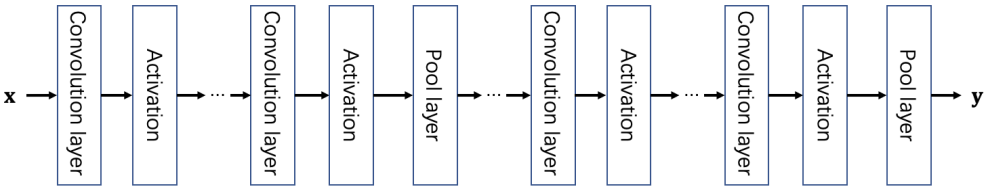

Fig. 2.9. The structure of convolutional neural networks.

In the CSP, convolution is often used to calculate the delay accumulation of a signal. We assume that a signal generator produces a signal x_t at each moment t, the information decay rate of time step k is w_k, i.e. the information at k step becomes w_k times of the information in step $k-1$. The signal y_t received at time t is the superposition of the information generated at the current time and the delayed information from the previous time. We can represent this as follows,

$$\begin{aligned} y_t &= w_t \cdot x_t + w_{t-1} \cdot x_{t-1} + ... + w_2 \cdot x_2 + w_1 \cdot x_1 \\ &= \sum_{k=1}^{t} w_k \cdot x_k \\ &= \mathbf{w} \otimes \mathbf{x}, \end{aligned} \tag{2.17}$$

where, $\otimes$ represents convolution operation. Here, we call $\mathbf{w}$ as convolution kernel, we also can see $\mathbf{w}$ as filter. This algorithm is the convolution operation. Then we introduce the convolution operation in neural networks. When we use a 5×5 matrices represents the input, and a 3×3 matrices to represent the convolution kernel (filter), we show the convolution operation in neural networks in Fig.2.10. In this example, the input is

$$\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -1 & 0 & -3 & 0 & 1 \\ 2 & 1 & 1 & -1 & 0 \\ 0 & -1 & 1 & 2 & 1 \\ 1 & 2 & 1 & 1 & 1 \end{bmatrix}. \tag{2.18}$$

The convolution kernel is

$$\mathbf{w} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.19}$$

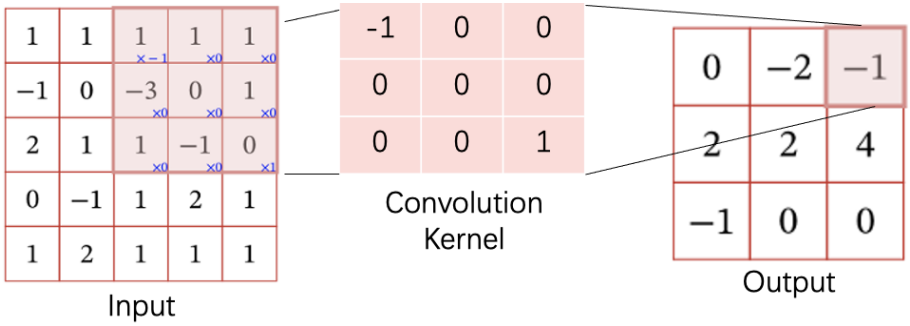

Fig. 2.10. The example of convolution kernel. The convolution kernel slides on the input matrix.

As we mentioned before, convolution kernel in convolutional neural networks has a receptive field. The convolution kernel only does convolution operation with elements in the receptive field. The size of the receptive field is the size of the convolution kernel. Therefore, in this example, the size of the receptive field is 3×3. At time t, elements of the receptive field is

$$\mathbf{x}_t = \begin{bmatrix} 1 & 1 & 1 \\ -3 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}, \tag{2.20}$$

therefore, the output of this time is

$$\begin{aligned} y_t &= \mathbf{x}_t \otimes \mathbf{w} \\ &= 1 \times (-1) + 1 \times 0 + 1 \times 0 + (-3) \times 0 + 0 \times 0 + 1 \times 0 + 1 \times 0 + (-1) \times 0 + 0 \times 1 \\ &= -1. \end{aligned} \tag{2.21}$$

In the convolution layer, parameter sharing is used to control the number of parameters. Parameter sharing base the following assumption:

Assumption 1 *If one feature is useful to compute at some spatial position* (x_1, y_1)*, then it should also be useful to compute at a different position* (x_2, y_2)*.*

Based on this assumption, the number of parameters can be significantly reduced. The size of the signal is $(H \times W \times D)$, for example, when the signal is a 256×256 RGB picture, the size of the signal is $(256 \times 256 \times 3)$. For each D, we call it depth slice, and only one filter (convolution kernel) is utilized to process it. This is the parameters sharing.

Generally, the size of the filter is smaller than the size of the input signal, Therefore, the convolution kernel slides in dimensions H and W in a depth slice. In the example in Fig.2.10, the step size of the slide is 1. We usually call the step size of the slide is stride. Finally, the output of each time consists of the output matrix

$$\mathbf{y} = \begin{bmatrix} 0 & -2 & -1 \\ 2 & 2 & 4 \\ -1 & 0 & 0 \end{bmatrix}. \tag{2.22}$$

We also call the output matrix $\mathbf{y}$ is a feature map.

According to the convolution operation, we can know that convolution kernel extracts features (a feature map) from input. Several convolution kernels make up a convolution layer. Therefore, the convolution layer extracts a set of features (a set of feature maps) from the input.

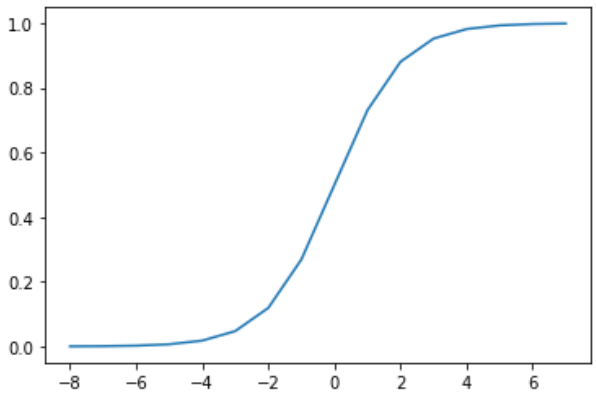

Fig. 2.11. The sigmoid function.

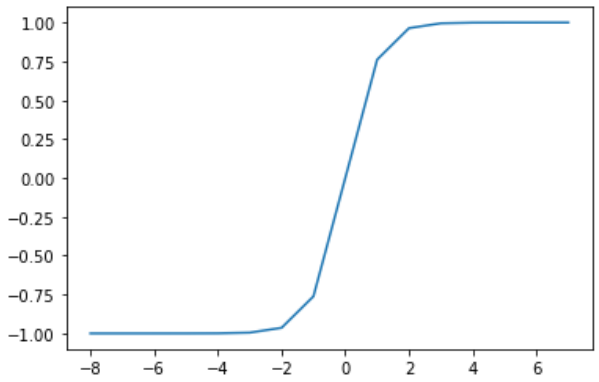

Fig. 2.12. The tanh function.

2.4.2 Activation layer

The convolution operation we mentioned before is the liner operation. However, as we all know, linear operations have many limits. Therefore, to process complex signals, we need non-linear convolution kernels. In convolutional neural networks, activation functions are utilized after the convolution layer to make the output become no-linear output. We call this layer an activation layer.

There are main three non-linear activation functions are utilized, i.e. sigmoid function (logistic sigmoid function), tanh function, and ReLU (Rectified Linear Unit) function.

sigmoid function The definition of sigmoid function is

$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}, \tag{2.23}$$

where, x is the input. Due to the definition, we can know that when x is small, the value of the output close to 0. When x is big, the value of the output close to 1. We show it in Fig.2.11. It is a smooth function that is easy to derive. However, Sigmoid has some disadvantages, i.e. prone to gradient vanishing, the function output is not zero-centered, power operations needs more compute source.

tanh function The definition of tanh function is

$$f(x) = tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}, \tag{2.24}$$

where, x is the input. We show it in Fig.2.12. Compare with sigmoid function, it solves the problem of zero-centered output, however, the problem of gradient vanishing and the problem of power operations remain.

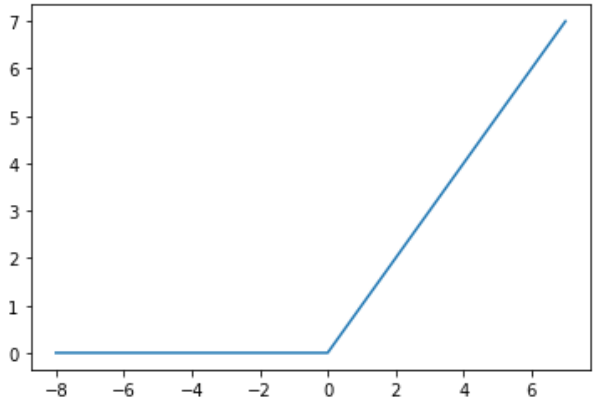

Fig. 2.13. The ReLU function.

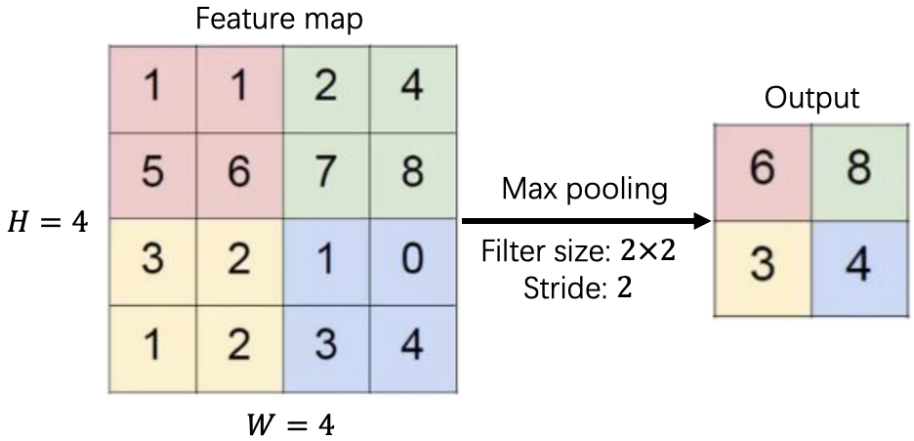

Fig. 2.14. The example of max-pooling operation. The size of feature map is 4×4 ($H \times W$), the filter size is 2×2, the stride is 2.

ReLU function The definition of ReLU function is

$$f(x) = ReLU(x) = max(0, x), \tag{2.25}$$

where, x is the input. We show it in Fig.2.13. Compare with sigmoid function and tanh function, ReLU function solves the problem of gradient vanishing (when $x \in [0, +\infty)$), and the calculation is very fast, it only needs to determine if the input is greater than 0. However, the output is not zero-centered.

In the convolutional neural networks, ReLU is the most commonly utilized activation function.

2.4.3 Pooling layer

A pooling layer is usually inserted periodically between successive convolutional layers. This layer gradually reduces the spatial size of the data, thus reducing the number of parameters in the network, making it need less computational resources and reducing overfitting.

In classic neural networks, max-pooling is widely utilized. It works on each feature map, changing the size of the feature map. An example of a max-pooling operation is shown in Fig. 2.14. However, it does not change the number of feature maps. For example, if we have a set of feature maps, the size of each feature map is $H_1 \times W_1$, the number of feature maps (Depth) is D_1. The size of max-pooling filter is $F \times F$, the stride is S. Therefore, the size

of output of max-pooling layer is

$$\begin{aligned} W_2 &= \frac{(W_1 - F)}{S} + 1, \\ H_2 &= \frac{(H_1 - F)}{S} + 1, \\ D_2 &= D_1. \end{aligned} \tag{2.26}$$

2.5 Graph Neural Networks

Recently, deep neural networks have been utilized in many fields and have gained remarkable success in these fields, such as detecting a target in images or videos, part segmenting target or scene segmenting in images or videos, and recognizing targets in images or videos [16, 17]. Both images and videos are regular date, classic deep learning approaches are challenging to process irregular data, owing to irregular data without a fixed sample order. Irregularly structured data could be processed more effectively by transforming those to graph-structured data. In this context, graph neural networks (GNNs) have received a lot of attention [15, 25, 44, 45].

According to the structure of the classical neural networks, constructing a graph convolution kernel is the core step in designing a graph neural network (GNN). The activation functions used in graph neural networks are the same as in classical neural networks. In this section, we first introduce the graph convolution layer. Then, we describe the graph pooling layer.

2.5.1 Graph Convolution layer

According to section 2.4, we can know that the convolution operation is to process and aggregate the local features, then propagate the result to the next layer. According to section 2.1, the local in a graph is a node and the neighbor nodes of this node. Therefore, the core idea of graph convolution operation is to aggregate the node-wise features of the neighbor nodes and process them, finally propagating them to the next layer. The most direct idea is to generalize convolution operations from other domains to the graph domain. Spectral methods and spatial methods are two common categories for advancements in this direction.

Spectral methods

As mentioned in section 2.4.1, convolution operations are usually utilized in CSP. Therefore, in the spectral methods, the theorem of GSP is used to design the graph convolution operation. The input is signal $\mathbf{x}$, and the steps of graph convolution operations are summarized as follows:

1 The signal is transformed to the spectral domain by GFT that we introduced in section 2.3.2, as follows:

$$F(\mathbf{x}) = \mathbf{U}^{\mathrm{T}}\mathbf{x} \tag{2.27}$$

where $F(\cdot)$ is the graph Fourier transform.

2 The convolution operation is conducted.

3 The result signal is transformed back using the inverse GFT that we introduced in section 2.3.2, as follows:

$$F(\mathbf{x})^{-1} = \mathbf{U}\mathbf{x} \tag{2.28}$$

where $F(\cdot)^{-1}$ is the inverse graph Fourier transform.

Here, $\mathbf{U}$ is the matrix of eigenvectors of the normalized graph Laplacian, the normalized graph Laplacian is defined as follows:

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \tag{2.29}$$

where, $\mathbf{D}$ is the degree matrix, $\mathbf{A}$ is the adjacency matrix of the graph. The normalized graph Laplacian is a real symmetric positive semidefinite matrix, then we can also define it as $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$, where $\mathbf{\Lambda}$ is a diagonal matrix of the eigenvalues. According to the convolution theorem [46], the graph convolution operation can be defined as follows:

$$\begin{aligned} \mathbf{g} \otimes \mathbf{x} &= F^{-1}(F(\mathbf{g}) \cdot F(\mathbf{x})) \\ &= \mathbf{U}(\mathbf{U}^{\mathrm{T}}\mathbf{g} \cdot \mathbf{U}^{\mathrm{T}}\mathbf{x}), \end{aligned} \tag{2.30}$$

Here, $\mathbf{U}^{\mathrm{T}}\mathbf{g}$ is the filter in the spectral domain. To simplify, we use a learnable matrix $\mathbf{g}_w$ to represent $\mathbf{U}^{\mathrm{T}}\mathbf{g}$, the basic function of the graph convolution operation is:

$$\mathbf{g} \otimes \mathbf{x} = \mathbf{U}\mathbf{g}_w\mathbf{U}^{\mathrm{T}}\mathbf{x}. \tag{2.31}$$

Therefore, the problem "How to do graph convolution ?" becomes the problem "How to design the learnable matrix $\mathbf{g}_w$?". In the follows, we introduce some typical spectral methods.

Spectral Network. This method is proposed by Bruna et al. [15]. In this method, a learnable diagonal matrix is utilized as the filter, i.e. $\mathbf{g}_w = \mathrm{diag}(\mathbf{w})$, $\mathbf{w}$ is the learnable parameter. Due to the method is not efficient to compute and the filter is non-spatially localized, Henaff et al. [47] seek to express spatial localization of filters in terms of their spectral multipliers. A parameterization with smooth coefficients are introduced.

ChebNet. Hammond et al. [18] propose that we can utilize a Kth order truncated extension of the Chebyshev Polynomial, i.e. $\mathbf{T}_k(x)$, to approximate the filter $\mathbf{g}_w$. Based on this method, Defferard et al. [32] propose the ChebNet. The graph convolution operation can be shown as follows:

$$\mathbf{g}_w \otimes \mathbf{x} \approx \sum_{k=0}^{K} w_k \mathbf{T}_k(\widetilde{\mathbf{L}})\mathbf{x}, \tag{2.32}$$

where, $\widetilde{\mathbf{L}} = \frac{2}{\lambda_{max}}\mathbf{L} - \mathbf{I}$, λ_{max} is the largest eigenvalue of $\mathbf{L}$, $\widetilde{\mathbf{L}} \in [-1, 1]$. $\mathbf{x}$ is a vector of Chebyshev coefficients. Chebyshev polynomials are defined as follows:

$$\begin{aligned} &\mathbf{T}_k(\mathbf{x}) = 2\mathbf{x}\mathbf{T}_{k-1}(\mathbf{x}) - \mathbf{T}_{k-2}(\mathbf{x}), \\ &\mathbf{T}_0(\mathbf{x}) = 1, \\ &\mathbf{T}_1(\mathbf{x}) = \mathbf{x}. \end{aligned} \tag{2.33}$$

From the definition, we can know that the kth order polynomial in the Laplacian represents the kth-hop neighbor nodes of a node. Therefore, we do not need to compute all eigenvectors of the Laplacian reducing the need of the compute source.

CayleyNet. Levie et. al [33] try to utilize Cayley polynomial to approximate the filter $\mathbf{g}_w$. It can be written as

$$\mathbf{g}_w \otimes \mathbf{x} \approx c_0\mathbf{x} + 2Re(\sum_{j=1}^{r} c_j(h\mathbf{L} - i\mathbf{I})^j(h\mathbf{L} + i\mathbf{I})^{-j}\mathbf{x}), \tag{2.34}$$

where, $Re(\cdot)$ is the real part of a complex number, $\mathbf{c} = (c_0, ..., c_r)$ is a vector of one real coefficient and r complex coefficients and h>0 is the spectral zoom parameter. CayleyNet shows that ChebNet can be seen as a special case of CaleyNet.

GCN. To further simplify the convolution operation, Kipf et al. [19] propose GCN. In GCN, 1th order Chebyshev polynomial is utlitized. Further, GCN assume that $\lambda_{max} \approx 2$. Therefore, the Eq. 2.32 can be changed as follows:

$$\begin{aligned} \mathbf{g}_w \otimes \mathbf{x} &\approx w_0\mathbf{x} + w_1(\mathbf{L} - \mathbf{I})\mathbf{x} \\ &= w_0\mathbf{x} - w_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x}, \end{aligned} \tag{2.35}$$

Furthermore, with parameter constraint $w = w_0 = w_1$, the expression can be written as:

$$\mathbf{g}_w \otimes \mathbf{x} \approx w(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{x}. \tag{2.36}$$

To solve the problems of the exploding / vanishing gradient in Eq.2.36, GCN introduces a renormalization trick as follows:

$$\begin{aligned} \mathbf{g}_w \otimes \mathbf{x} &\approx w(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{x} \\ &= w\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{x}, \end{aligned} \tag{2.37}$$

where, $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\widetilde{\mathbf{D}}_{ii} = \sum_j \widetilde{\mathbf{A}}_{ij}$. Finally, the GCN can be written as:

$$\mathbf{H} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}, \tag{2.38}$$

where, $\mathbf{X} \in \mathbb{R}^{N \times F}$ is the input matrix, $\mathbf{W} \in \mathbb{R}^{F \times F^{'}}$ is the learnable parameters, $\mathbf{H} \in \mathbb{R}^{N \times F^{'}}$ is the convolved matrix and be propagated to next layer. N is the number of the nodes, F and $F^{'}$ is the dimensions of the input and the output, respectively.

Due to GCN utilizes 1th order Chebyshev polynomial, it just utilize 1-hop neighbor nodes a node in a graph. Therefore, it do not need to calculate Laplacian matrix, the complexity of graph convolution can be reduced. GCN becomes the most wildly utilized method of the Spectral methods.

Spatial methods

Due to spectral methods needing to transform the graph to spectral space and transform the result to the spatial space, these make methods complex. Therefore, how to design a graph convolution in spatial space attaches a lot of attention. Spatial methods define graph convolution directly on the graph topology. The main challenge of spatial graph convolution operation is to process nodes with differently sized neighborhoods and keep the local invariant.

The steps of the spatial graph convolution are as follows:

1 Collects and aggregates features on the surrounding neighbors;

2 Merges features into the target node.

In the follows, we introduce typical spatial methods.

Neural FPs. Duvenaud et al. proposed Neural FPs [48]. In this method, different weight matrices are utilized for nodes with different degrees. The graph convolution operation can be written as follows:

$$\begin{aligned} \mathbf{t} &= \mathbf{h}_v + \sum_{u \in N(v)} \mathbf{h}_u, \\ \mathbf{h}_v &= \sigma(\mathbf{t}\mathbf{W}_{|N(v)|}). \end{aligned} \tag{2.39}$$

where, $\mathbf{h}_v$ is the node-wise features of target node v, $\mathbf{h}_u$ is the node-wise features of neighbor node u. $N(v)$ is the set of neighbor nodes of the target node v, $|N(v)|$ is the number of neighbor nodes of the target node v (the degree of the target node v). $\mathbf{W}_{|N(v)|}$ is the learnable weight matrix for nodes with degree $|N(v)|$. $\sigma(\cdot)$ is non-linear activation function.

The shortage of the Neural FPs is that it can not be utilized for large-scale graphs with more node degrees.

GraphSAGE. GraphSAGE (Hamilton et al. [29]) is a general framework. It generates embeddings by sampling and aggregating node-wise features from the local neighborhood of the target node, then merging it with the node-wise features of the target node. It can be written as follows:

$$\begin{aligned} \mathbf{h}_{N(v)} &= \mathrm{AGG}(\{\mathbf{h}_u, u \in N(v)\}), \\ \mathbf{h}_v &= \sigma(\mathbf{W} \cdot \mathrm{cat}(\mathbf{h}_v, \mathbf{h}_{N(v)})). \end{aligned} \tag{2.40}$$

where $\mathbf{W}$ is a learnable weight matrixes, AGG is the aggregation operation, $\mathrm{cat}(\cdot)$ is the concatenation operation. Instead of utilizing all neighbor nodes of the target node, GraphSAGE uniformly samples a fixed-size set of neighbor nodes. For AGG operation, GraphSAGE suggests three aggregation operations:

A. Mean aggregator;

B. LSTM aggregator;

C. Pooling aggregator.

GraphSAGE with a mean aggregator can be seen as a special version of GCN. The LSTM aggregator needs a specified order of the nodes.

Attention-based Spatial methods

Attention mechanism has been widely utilized in many tasks, such as machine translation [49–51], machine reading [52], and so on. Therefore, introducing an attention mechanism to graph convolution is also considered.

GAT. Graph attention network [30] introduce attention mechanism to graph convolution operation. The hidden embedding of node v can be obtained as follow:

$$\begin{aligned} \mathbf{h}_v &= \sigma(\sum_{u \in N(v)} a_{vu} \mathbf{W} \mathbf{h}_u), \\ a_{vu} &= \frac{exp(LeakyReLU(\mathbf{a}^{\mathrm{T}} [\mathbf{W}\mathbf{h}_v || \mathbf{W}\mathbf{h}_u]))}{\sum_{i \in N(v)} exp(LeakyReLU(\mathbf{a}^{\mathrm{T}} [\mathbf{W}\mathbf{h}_v || \mathbf{W}\mathbf{h}_i]))}. \end{aligned} \tag{2.41}$$

where $\mathbf{a}$ is the weight vector of a single-layer MLP and $\mathbf{W}$ is the weight matrix associated with the linear transformation that is applied to each node. $||$ is the concatenation operation.

GAT also utilizes multi-head attention. It uses K independent attention head to compute the hidden states and concatenates them or compute the average of them. Output representations are shown as follows:

$$\begin{aligned} \mathbf{h}_v &= ||_{k=1}^{K} \sigma(\sum_{u \in N(v)} a_{vu}^k \mathbf{W} \mathbf{h}_u), \\ \mathbf{h}_v &= \sigma(\frac{1}{K} \sum_{k=1}^{K} \sum_{u \in N(v)} a_{vu}^k \mathbf{W} \mathbf{h}_u). \end{aligned} \tag{2.42}$$

Here the result of the kth attention head a_{vu}^k is the normalized attention coefficient. The attention architecture has three properties:

1 The operation is effective because the computation of the node-neighbor pairs is parallelized;

2 It can be used on graph nodes with various degrees by giving neighbors arbitrary weights;

3 It is easily applicable to the issues with inductive learning.

2.5.2 Graph Pooling layer

As mentioned in section 2.4, convolution layers are usually followed by a pooling layer. The pooling layer can get more general features and reduce the compute source. Large-scale and complicated graphs usually have rich hierarchical structures that are crucial for node classification and graph classification tasks. Inspired by the pooling layer in classical neural networks, a lot of works [53, 54] try to design a graph pooling layer. In this section, we introduce graph pooling methods. Graph pooling methods include two kinds:

1 Direct pooling methods;

2 Hierarchical pooling methods.

Direct pooling methods

Direct pooling methods learn graph-level representations directly from nodes with different node selection strategies.

Sample Node Pooling. In these methods, the max/mean/sum pooling operations, which are utilized in classical neural networks, are directly applied on node-wise features to get a global graph representation. These methods are also wildly utilized in graph neural networks.

Set2set. MPNN utilizes this method [55] to get the graph representations. Set2set employs an LSTM-based approach to construct an order-invariant representation after a specified number of steps when dealing with the unordered set $T = \{(\mathbf{h}_v^{\mathrm{T}}, \mathbf{x}_v)\}$, where $\mathbf{x}_v$ is the input of the MPNN of the node v, $\mathbf{h}_v$ is the hidden features of the node v.

SortPooling. Prior to feeding the sorted embeddings into CNNs to produce the representation, SortPooling [56] sorts the node embeddings based on the structural roles of the nodes.

Hierarchical pooling methods

Here, we introduce methods that follow a hierarchical graph pooling pattern and learn graph representations by layers.

Graph Coarsening. Early methods are usually based on graph coarsening algorithms. In these methods, spectral clustering algorithms are directly utilized, but due to spectral clustering algorithms need eigendecomposition, they are may inefficient. Dhillon et al. [57] presents Graclus which provides a faster way to cluster nodes.

ECC. In this method [58], recursively downsampling operation are utilized as graph pooling operation. In the downsampling method, the sign of the largest eigenvector of the Laplacian is used to divide the graph into two parts.

DiffPool. In each layer, DiffPool [59] utilize a learnable hierarchical clustering module to learn an assignment matrix $\mathbf{S}$. It can be written as follows:

$$\begin{aligned} \mathbf{S} &= \mathrm{softmax}(\mathrm{GNN}(\mathbf{A}, \mathbf{H})), \\ \mathbf{A} &= \mathbf{S}^{\mathrm{T}}\mathbf{A}\mathbf{S}, \end{aligned} \tag{2.43}$$

where $\mathbf{H}$ is the set of node-wise features, $\mathbf{A}$ is the adjacent matrix, $\mathrm{GNN}(\cdot)$ is the graph convolution operation.

gPool. gPool is proposed in Graph U-net [60]. It uses a learnable project vector to learn projection scores for each node, and select node with top k scores. Compared with DiffPool, gPool just use a learnable vector instead of matrix at each layer, thus it is more efficient. It can be written as follows:

$$\begin{aligned} \mathbf{score} &= \frac{\mathbf{H}\mathbf{P}}{l2norm(\mathbf{P})}, \\ idx &= \text{top-rank}(\mathbf{score}, k), \\ \mathbf{A} &= \mathbf{A}_{idx,idx}, \end{aligned} \tag{2.44}$$

where $\mathbf{H}$ is the set of node-wise features, $\mathbf{P}$ is the learnable vector, $l2norm(\cdot)$ is the l2 normalization, top-rank find the index of the top k scores. However, we can know that the projection procedure does not consider the graph structure.

EigenPooling. EigenPooling [61] utilizes the node-wise features and local structure jointly. It utilizes the local graph Fourier transform to get subgraph information. Therefore, it can suffers from the drawback of graph eigendecomposition.

SAGPool. SAGPool [62] is also utilize the node-wise features and topology jointly. It employs a self-attention-based technique with a tolerable level of temporal and spatial complexity.

Chapter 3

Structural features in feature space for structure-aware graph convolution

3.1 Introduction

Deep neural networks have been remarkably successful at identifying, segmenting, and comprehending regularly structured data, such as imagess and videos, thanks to massive datasets and advances in computer power [15–17]. However, in the actual world, there are a lot of irregular data without a fixed sample order that are challenging for conventional deep learning techniques to analyze. These examples include social network opinions, 3D point clouds, and traffic network passenger counts. Irregular data can be processed effectively by being transformed into graph-structured data. Research on graph neural networks (GNNs) has drawn a lot of interest because of this [25].

As we discussed in chapter 2, graph convolutions (GCs), also known as graph filters, are frequently used in existing GNNs. The primary concept of GC algorithms is to integrate the aggregated data with that of the target node by first iteratively aggregating features from neighbors [25, 26, 63]. Existing methods, however, primarily use node-wise features of the one-hop neighborhood in one step of GC [19, 29, 30]. This could lead to the over-smoothing [25, 28]. Furthermore, over-smoothing influences the performance of GNNs. Improving the expressive of GCs can reduce the possibility of over-smoothing [64]. One limitation of the expressive of GCs is the loss of the structural data of immediate neighbors of the target node, particularly in the feature space. In other words, the expressive of GCs may be further enhanced if we can effectively employ the comprehensive structure data as well as node-wise features. Furthermore, the performance of GNNs may also be improved.

To utilize the structural information of immediate neighbors of the target node in the feature space, we propose three structural features. The following structural features are computed in our proposed SAGConv: feature angle (fa_{uv}), feature distance (fd_{uv}), and relational embedding (re_{uv}). We then combine structural features with node-wise features. After that, they are introduced into the GC process.

The following is a summary of our contributions:

(1) We can get the specific structural details of the surrounding neighboring nodes in feature space by specifying the three structural features. Compared to existing methods, it enables the processed nodes to contain richer information. In other words, the expressive of GC is improved.

(2) We build two GNNs for graph and node classification tasks based on SAGConv. We

take into consideration 3D point clouds while classifying graphs. We incorporate SAGConv along with a layer of PointNet++ [65] and a graph pooling operation into our 3D point cloud classification network. Through experiments on the ModelNet [66] dataset, we show that our method has higher classification accuracy than other methods.

We cascade three SAGConv layers with a fully connected layer to classify nodes on a graph. We show that the performance of our method is similar to other methods currently in use and created especially for the node classification of the Cora dataset [67].

3.2 Preliminaries

As we described in chapter 2, there are two ways to do GCs: spectral methods and spatial methods. Most spectral GCs assume that a fixed graph is used during training and testing since spectral methods need eigendecomposition. The opposite of spectral methods are spatial methods.

Spatial GCs only perform GC in one-hop neighborhood of a target node in spatial space. As a result, spatial methods may be more effective for dynamic graphs and require less computing power. For these reasons, we use GraphSAGE [29], a general framework that was described in chapter 2, as our skeleton. The following is a representation of the spatial graph convolution operation:

$$h'_v = \text{integration}(h_{N(v)}, h_v), v \in V \tag{3.1}$$

where

$$h_{N(v)} = \text{aggregation}(h_u), u \in N(v) \tag{3.2}$$

in which V is the set of nodes in the input graph, h'_v is the updated node-wise feature of the target node v. Furthermore, $h_{N(v)}$ is the aggregated node-wise feature at $N(v)$. aggregation$(\cdot)$ is the aggregation operation, integration$(\cdot)$ is the integration operation. As we described before, existing GCs only utilize node-wise features. Therefore, they may occur over-smoothing [25, 28]. Furthermore, over-smoothing may influence the performance of GNNs. An effective strategy to reduce the probability of over-smoothing is to improve the expressive of GC [64].

Therefore, in our SAGConv, we introduce the proposed structural features into the aggregation$(\cdot)$ operation to improve the expressive of GC. Then we update the node-wise features of the target node.

3.3 SAGConv

In this section, we introduce the SAGConv. It is depicted in Fig.3.1. As mentioned earlier, our objective is to utilize the structural information of the surrounding neighborng nodes in the feature space during a spatial graph convolution. To achieve this, the SAGConv contains three ingredients:

A. Structural Features;

B. Neighborhood Average Aggregation;

C. Integration.

Its details are shown in Algorithm 1, and we sequentially introduce the three components.

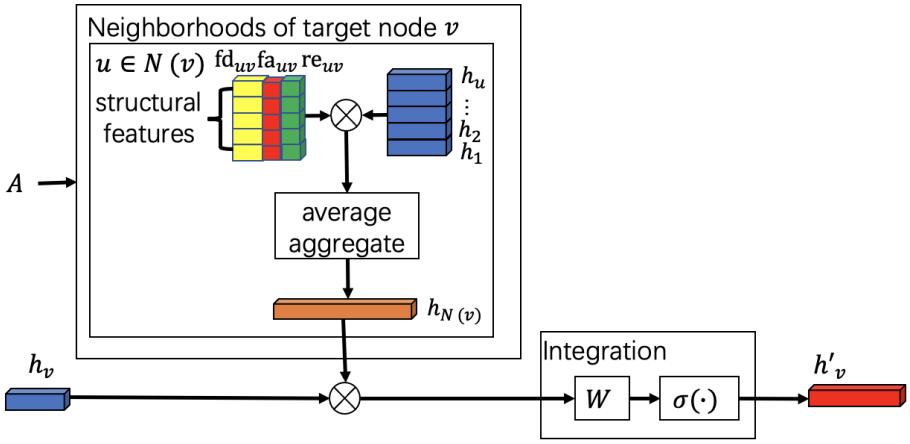

Fig. 3.1. SAGConv. $\otimes$ is the concatenation operation, W is the learnable parameters, $\sigma(\cdot)$ is the non-liner activation function (RaLU).

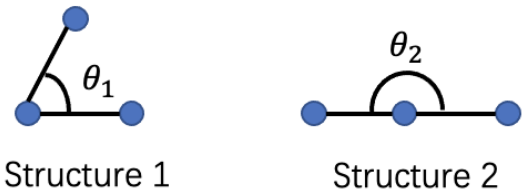

Fig. 3.2. Angle of space. We can differentiate Structure 1 and Structure 2 using the angle θ_1 and the angle θ_2.

3.3.1 Structural Features

In this section, we introduce our structural features. These structural features are utilized to obtain the detailed structural information on the surrounding nodes in feature space.

For an undirected graph $G = (V, E)$, V is a set of nodes and E is a set of edges. h_v represents features on the node v, $v \in V$. Features on the node u connected to v are also defined as h_u, $u \in N(v)$, where $N(v)$ is the set of neighboring nodes of v.

Feature angle

To describe a structure of a space, angles are very important. For example, in Fig. 3.2, we can differentiate Structure 1 and Structure 2 using the angle θ_1 and the angle θ_2. Therefore, we also want to utilize angle to describe the structure of the neighborhood of a node in the feature space.

To calculate angles, we first calculate a set of vectors pointing from target node v to the surrounding neighbor nodes u, as follows:

$$\mathcal{F}_{N(v)} = \{g_{uv} := h_u - h_v\}, u \in N(v), \tag{3.3}$$

The most natural idea is to calculate the cosine of the angle between a vector and others, as follows:

$$\begin{aligned} \mathrm{fa}_{uv} &= \{\cos(\theta_{1u}), \cos(\theta_{2u}), ..., \cos(\theta_{uu}), ..., \cos(\theta_{|N(v)|u})\} \\ &= \{\frac{g_{1v} \cdot g_{uv}^{\mathrm{T}}}{\|g_{1v}\| \cdot \|g_{uv}\|}, \frac{g_{2v} \cdot g_{uv}^{\mathrm{T}}}{\|g_{2v}\| \cdot \|g_{uv}\|}, ..., \frac{g_{uv} \cdot g_{uv}^{\mathrm{T}}}{\|g_{uv}\| \cdot \|g_{uv}\|}, ..., \frac{g_{|N(v)|v} \cdot g_{uv}^{\mathrm{T}}}{\|g_{Nv}\| \cdot \|g_{uv}\|}\}_{g_{uv} \in \mathcal{F}_{N(v)}} \end{aligned} \tag{3.4}$$

where, $|N(v)|$ is the number of the neighboring nodes. This process is shown in Fig.3.3.

Algorithm 1 SAGConv spatial graph convolution (i.e., forward propagation) algorithm

Input: graph $G = (V, E)$; input features $\{h_v, v \in V\}$; weight matrices W; non-linear activation function $\sigma(\cdot)$; the neighborhood of a node $N(\cdot)$; MLP layer $\mathrm{MLP}(\cdot)$; channel-wise max-pooling $\mathrm{MaxPool}(\cdot)$; concatenation operation $\mathrm{cat}(\cdot)$; transposition $\cdot^{\mathrm{T}}$; C is the number of channels of a node-wise feature.

Output: Convolved features z_v for all $v \in V$

for $v \in V$ **do**
  $\mathcal{F}_{N(v)} = \{g_{uv} := h_u - h_v\}$
  $g_b = \mathrm{MaxPool}(\{\sigma(\mathrm{MLP}(g_{uv}))\})$
  $\mathrm{fa}_{uv} = \cos(\theta_u) = \dfrac{g_{uv} \cdot g_b^{\mathrm{T}}}{\|g_{uv}\| \cdot \|g_b\|}, g_{uv} \in \mathcal{F}_{N(v)}$
  $\mathrm{fd}_{uv} = [|\mathrm{h}_{u1} - \mathrm{h}_{v1}|, ..., |\mathrm{h}_{uC} - \mathrm{h}_{vC}|]^{\mathrm{T}}$
  $\mathrm{re}_{uv} = \sigma(\mathrm{MLP}(h_u - h_v)), u \in N(v)$
  $h_{N(v)} = \frac{1}{|N(v)|} \sum_{u \in N(v)} \mathrm{cat}(h_u, \mathrm{fa}_{uv}, \mathrm{fd}_{uv}, \mathrm{re}_{uv})$
  $h'_v = \sigma\left(W \cdot \mathrm{cat}\left(h_v, h_{N(v)}\right)\right)$
end for
$z_v = h'_v, \forall v \in V$
return z_v

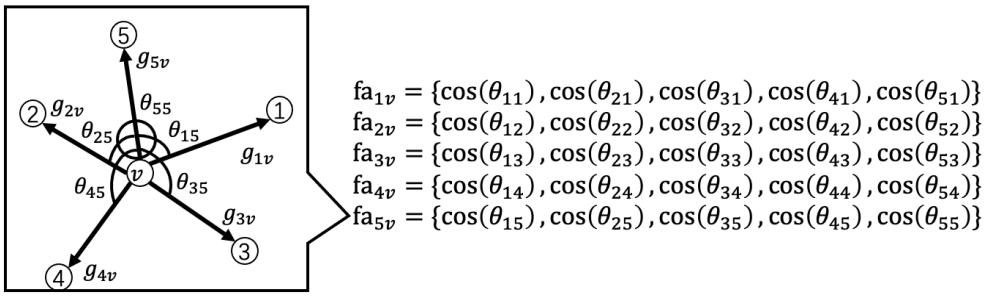

Fig. 3.3. The most natural idea is to calculate the cosine of the angle between a vector and others.

Then, aggregate the fa_{uv} as follows:

$$\mathrm{fa}_{uv} = \mathrm{aggregate}(\mathrm{fa}_{uv}), g_{uv} \in \mathcal{F}_{N(v)}, \tag{3.5}$$

where $\mathrm{aggregate}(\cdot)$ is the aggregation operation. The aggregation operation can be chosen as summation, average, and others.

However, we can know that if the number of neighboring nodes is large or the graph has a large scale, this process needs a lot of computing sources. Therefore, we consider finding a reference axis of a neighborhood of a target node in the feature space, then we calculate the cosine of the angle between a vector and the reference axis. Here, we use two methods to calculate the reference axis, we name the reference axis as the basis vector (g_b).

Non-learn method The first method is that we find the basis vector from the $\mathcal{F}_{N(v)}$ directly. Here, we propose three basis vectors.

1). ***Nearest basis vector*** We think that the neighboring node, which is nearest to target node v, is the most similar to target node v. Therefore the vector between this neighboring node and the target node can be used as basis vector.

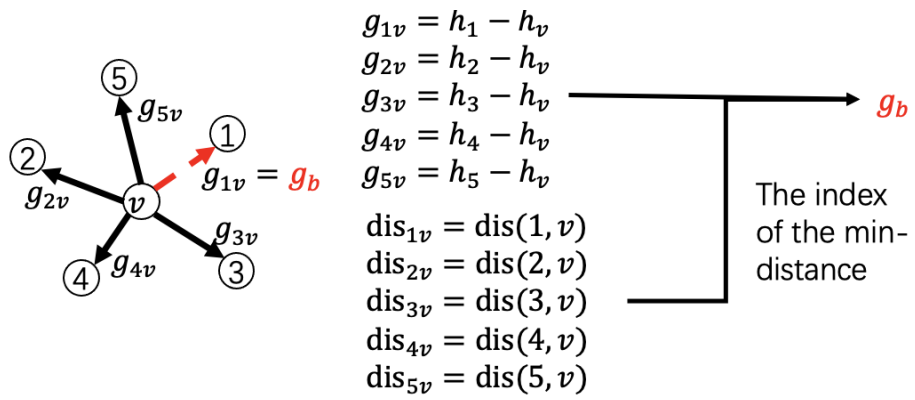

Fig. 3.4. The nearest base vector g_b. The numbers in the black circle are the node indices. dis$(\cdot)$ is function that calculates the distance between two nodes. dis_{uv} is the distance between neighboring node u and target node v.

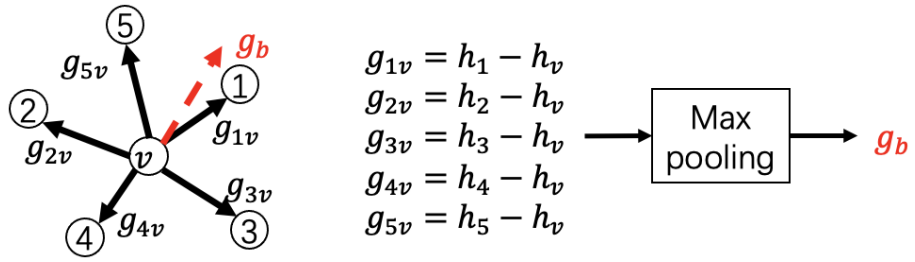

Fig. 3.5. The max base vector g_b. The numbers in the black circle are the node indices. MaxPool$(\cdot)$ is the channel-wise max-pooling operation.

We first find the neighboring node that is nearest to the target node v. Then we get the index of this neighboring node, as follows:

$$\mathrm{idx} = \mathrm{rank}(\mathrm{dis}(u, v)), u \in N(v), \tag{3.6}$$

where idx is the index of the neighboring node, which is nearest to target node v. rank$(\cdot)$ is the neighboring nodes ranking operation that returns index of the nearest neighboring node. dis$(\cdot)$ is the distance between the neighboring node u to target node v. Finally, we utilize the idx find the vector in $\mathcal{F}_{N(v)}$, and we use this vector as the basis vector as follow:

$$g_b = \mathcal{F}_{N(v)}[\mathrm{idx}]. \tag{3.7}$$

We show this method in Fig.3.4.

2). ***Max basis vector*** The second basis vector is the channel-wise max-pooling of the set $\mathcal{F}_{N(v)}$. It can be represented as follows:

$$g_b = \mathrm{MaxPool}(\mathcal{F}_{N(v)}). \tag{3.8}$$

where MaxPool$(\cdot)$ is the channel-wise max-pooling operation. We show it in Fig.3.5.

3). ***Average basis vector*** The third basis vector is the channel-wise average of the set $\mathcal{F}_{N(v)}$. It can be summarized as follows:

$$g_b = \mathrm{Average}(\mathcal{F}_{N(v)}). \tag{3.9}$$

where Average$(\cdot)$ is the channel-wise average operation. It is described in Fig.3.6.

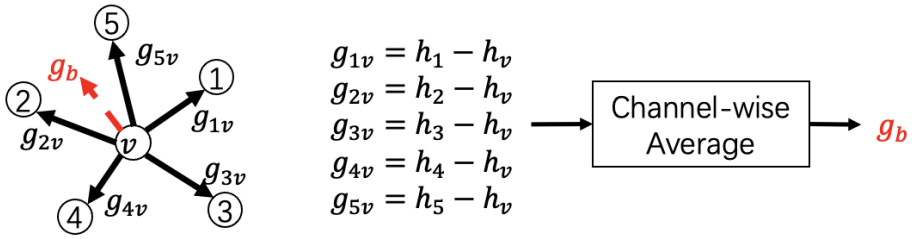

Fig. 3.6. The average base vector g_b. The numbers in the black circle are the node indices. Average(·) is the channel-wise average operation.

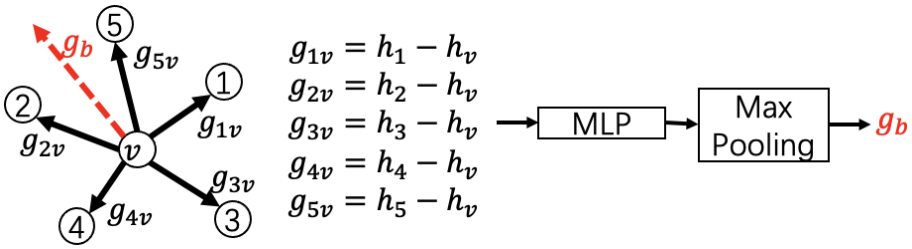

Fig. 3.7. Learning a base vector g_b. The numbers in the black circle are the node indices.

Learn method The second method is that we learn the base vector from the set of vectors pointing from target node v to the surrounding neighbor nodes u, i.e., $\mathcal{F}_{N(v)}$. In details, we utilize a MLP (Multi-layer Perceptron) and a channel-wise MaxPool layer to learn the reference axis from $\mathcal{F}_{N(v)}$, . In addition, a channel-wise AveragePool layer, sum operation or others also can be utilized to get the basis vector g_b. This propose can be summarized as follows:

$$g_b = \text{MaxPool}(\{\sigma(\text{MLP}(g_{uv}))\}_{g_{uv} \in \mathcal{F}_{N(v)}}), \tag{3.10}$$

where MLP(·) is the MLP layer, $\sigma(\cdot)$ is a nonlinear activation function (ReLU), MaxPool(·) is the channel-wise MaxPool layer. We show an example of this process in Fig.3.7.

In our SAGConv, we use the learn method to get our base vector. Finally, we compute the cosine of the angle between g_{uv} and g_b as follows:

$$\begin{aligned} \text{fa}_{uv} &= \cos(\theta_u) \\ &= \frac{g_{uv} \cdot g_b^{\mathrm{T}}}{\|g_{uv}\| \cdot \|g_b\|}, g_{uv} \in \mathcal{F}_{N(v)}. \end{aligned} \tag{3.11}$$

The example is depicted in Fig.3.8. We think our process to calculate the feature angle can reduce the need for computing sources.

Feature distance

Distance is another important value to describe a structure of a space. In the machine learning, distance is also important and widely utilized. In mathematic, there are many distances. The most widely utilized distance is Euclidean distance. Euclidean distance between node $x = (x_1, x_2, ..., x_d)$ and node $y = (y_1, y_2, ..., y_d)$ is calculated as follow:

$$\text{E}(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2} \tag{3.12}$$

where, d is the dimension of the feature space.

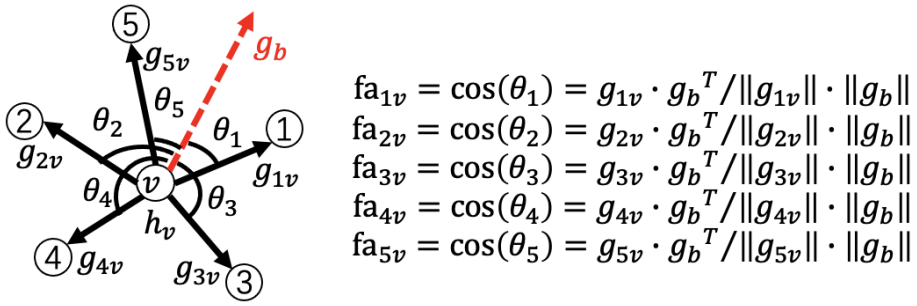

Fig. 3.8. Example of our feature angle.

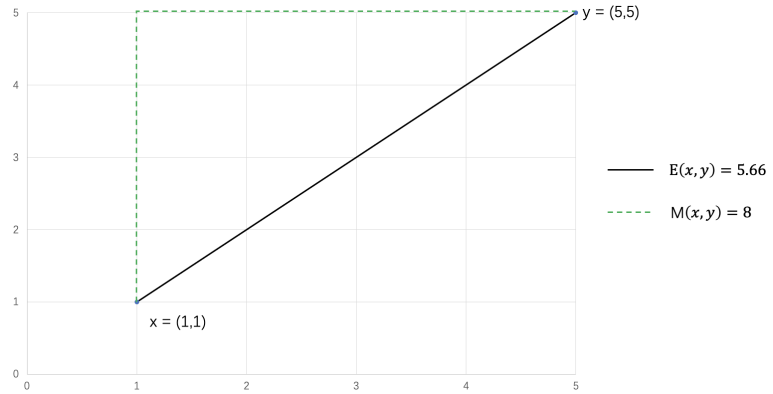

Fig. 3.9. The different of Euclidean distance and Manhattan distance. The Euclidean distance between node x and y is $\mathrm{E}(x,y) = \sqrt{\sum_{i=1}^{2}(x_i - y_i)^2} = 5.66$. The Manhattan distance between node x and y is $\mathrm{M}(x,y) = \sum_{i=1}^{2}|x_i - y_i| = 8$.

The another distance uaually utilized is the Manhattan distance. Manhattan distance is proposed by Hermann Minkowski in the 19th century. It is a geometric distance used in geometric metric space to indicate the sum of the absolute axis distances of two points on a standard coordinate system. The Manhattan distance between node $x = (x_1, x_2, ..., x_d)$ and node $y = (y_1, y_2, ..., y_d)$ is calculated as follow:

$$\mathrm{M}(x,y) = \sum_{i=1}^{d}|x_i - y_i| \tag{3.13}$$

where, d is the dimension of the feature space. The different of Euclidean distance and Manhattan distance is summarized in Fig.3.9.

However, both Euclidean distance and Manhattan distance may not describe the details of the structure in the feature space. Therefore, to describe the structure in the feature space more detailed, in our method, the distance between neighboring nodes u and target node v in the feature space is represented by the absolute difference between the elements of h_u and h_v. It can be represented as follows:

$$\mathrm{fd}_{uv} = [|\mathrm{h}_{u1} - \mathrm{h}_{v1}|, ..., |\mathrm{h}_{uC} - \mathrm{h}_{vC}|]^{\mathrm{T}} \tag{3.14}$$

The example is shown in Fig.3.10.

Relational embedding

The strength of the impact of one-hop neighboring nodes u on the target node v is represented by relational embedding. We have many ways to represent it. For example, we can use the

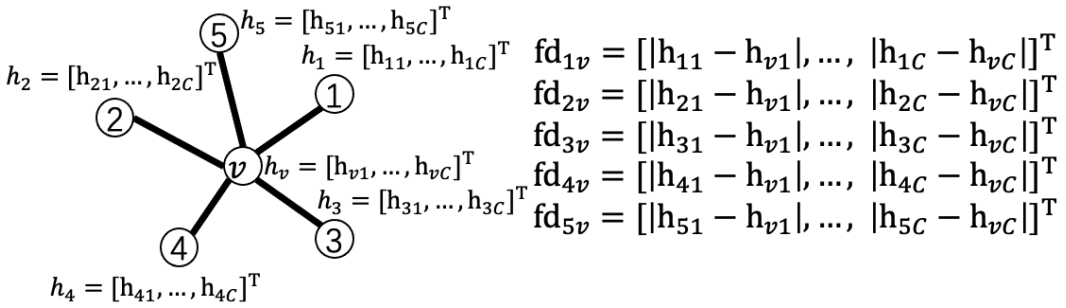

Fig. 3.10. Example of our feature distance.

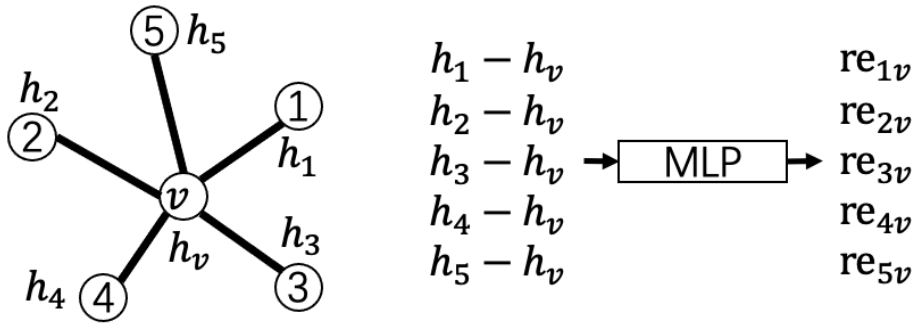

Fig. 3.11. Example of our relational embedding.

Euclidean distance to calculate the relational embedding as follows:

$$\mathrm{re}_{uv} = \frac{1}{1 + \mathrm{E}(u, v)}, u \in N(v), \tag{3.15}$$

where $\mathrm{E}(u, v)$ is the Euclidean distance between the neighboring node u and the target node v. We also can utilize a Gaussian Kernel to calculate the relation embedding as follows:

$$\begin{aligned} \mathrm{re}_{uv} &= \exp(\frac{-\mathrm{E}(u, v)^2}{\sigma^2}) \\ \sigma &= \frac{\sum_u^{|N(v)|} \mathrm{E}(u, v)}{|N(v)|}, u \in N(v) \end{aligned} \tag{3.16}$$

Counterpart to these, we learn it from the difference between h_v and h_u as follows:

$$\mathrm{re}_{uv} = \sigma(\mathrm{MLP}(h_u - h_v)), u \in N(v). \tag{3.17}$$

re_{uv} is depicted in Fig.3.11. We think learned relational embedding can better describe the strength of the impact of one-hop neighboring nodes u on the target node v.

3.3.2 Neighborhood Average Aggregation and Integration

As we described in chapter 2, GraphSAGE use three type of aggregator i.e. Average aggregator, LSTM aggregator and Pooling aggregator. Here, in our method, we use average aggregator. The aggregator can be represented as follows:

$$h_{N(v)} = \frac{1}{|N(v)|} \sum_{u \in N(v)} \mathrm{cat}(h_u, \mathrm{fa}_{uv}, \mathrm{fd}_{uv}, \mathrm{re}_{uv}), \tag{3.18}$$

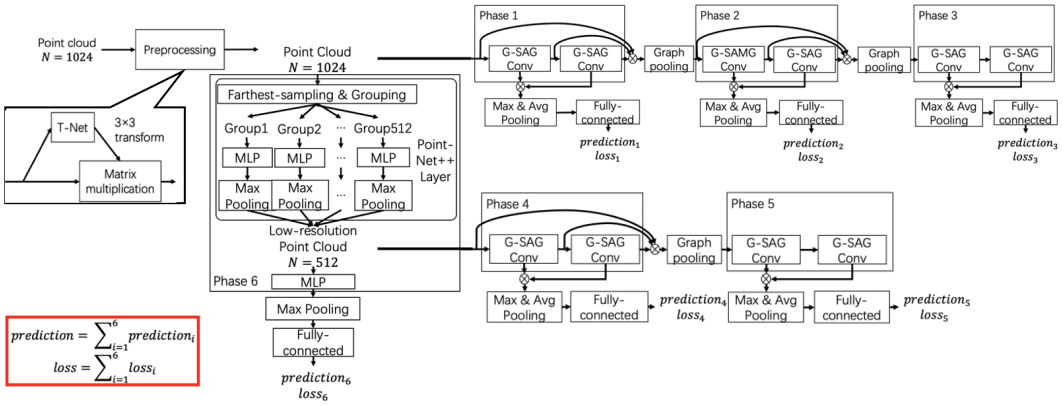

Fig. 3.12. Architecture of the 3D point cloud classification network where $\otimes$ represents the concatenation operation. The model adds up the losses and predictions from the different phase to obtain the overall classification loss and final prediction.

where $\mathrm{cat}(\cdot)$ denotes the concatenation operation. Finally, we integrate $h_{N(v)}$ with the feature on v as follows:

$$h'_v = \sigma\left(W \cdot \mathrm{cat}\left(h_v, h_{N(v)}\right)\right), \tag{3.19}$$

where W is a learnable matrix, and h'_v is convolved features on v. Hereafter, we denote the set of this operations as $h'_v := \mathrm{SAGConv}(h_V)$.

3.4 Implementation

In this section, we introduce GNNs using SAGConv. We propose a 3D point cloud classification network based on SAGConv in section 3.4.1. A 3-layer SAGConv-based graph node classification network is also shown in section 3.4.2.

3.4.1 3D Point Cloud Classification

Fig.3.12 illustrates the overall structure of the SAGConv-based 3D point cloud classification network. Suppose that the input point cloud is given by $X = \{x_i\}_{i=1}^{N}$ where N is the number of points.

Description

We preprocess the point cloud using the same transformation module as PointNet [68], called T-Net, to minimize rotational effects. In addition, we build a low-resolution point cloud using a layer of PointNet++ [65]. Both global and local information of the point cloud can be easily extracted because to the multi-resolution structure.

The specifics of the building blocks created expressly for point cloud classification are described in the sections that follow.

T-Net

This module is utilized to reduce rotational effects on the point cloud [68]. It is a mini-PointNet. The input of this module is the raw point cloud. These processes are summarized

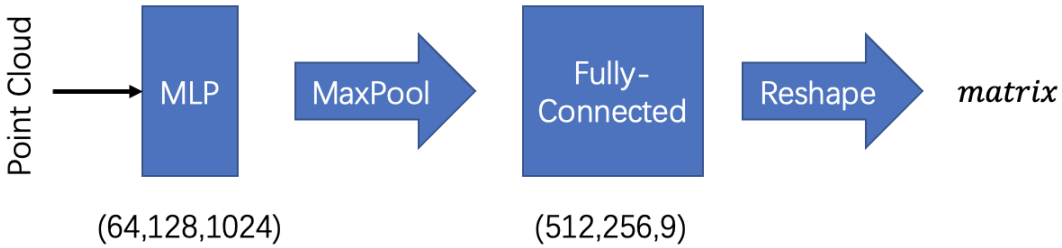

Fig. 3.13. The architecture of the T-Net. The output size of layers in MLP is $(64, 128, 1024)$. The output size of layers in fully-connected layer is $(512, 256, 9)$.

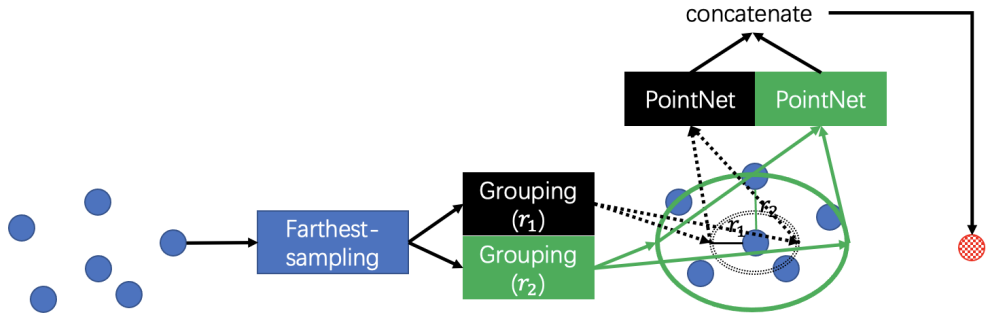

Fig. 3.14. The example of our PointNet++ layer. In this example, Farthest sampling is utilized to pick one center point. Then, for the center point, two radius r_1 and r_2 are used to get the neighboring points. Each group of points of a center point is processed using PointNet. Finally, the outputs are concatenated as the representation of the local of the center point.

as follows:

$$\begin{aligned} H &= \text{MLP}(X), \\ H &= \text{MaxPool}(H), \\ H &= \text{Fully-Connected}(H), \\ matrix &= \text{Reshape}(H) \end{aligned} \tag{3.20}$$

where X is the input, i.e., the point cloud. MLP$(\cdot)$ is a MLP (Multi-layer Perceptron), MaxPool$(\cdot)$ is a channel-wise MaxPool layer, Fully-Connected$(\cdot)$ is a fully connected layer, Reshape$(\cdot)$ is a reshape operation to make the shape of H to 3×3. The output matrix is initialized as an identity matrix. The architecture of T-Net is shown in Fig.3.13.

PointNet++ layer

PointNet++ is a hierarchical architecture Pointnet. The example of our PointNet++ layer is shown in Fig.3.14. Compared with PointNet, PointNet++ can capture local structure of point cloud using sampling layer and grouping layer. Here, we first introduce the PointNet++ layer. PointNet++ layer contains three layers, i.e. sampling layer, grouping layer and PointNet layer.

Sampling layer Given a point cloud $X = \{x_i\}_{i=1}^{N}$, the farthest-sampling is utilized to get a subset of points $X_s = \{x_{ij}\}_{j=1}^{m}$. The point x_{ij} is the furthest away from the rest points (in terms of metric distance) from the set $X_s = \{x_{ij}\}_{j=1}^{m}$. Given the same number of centroids as random sampling, farthest-sampling provides better coverage of the complete point set than random sampling.

Grouping layer The input of this layer is a point set with size $N \times 3$ and the sub point set with size $N' \times 3$, which is sampled by sampling layer. The output of this layer is point sets of size $N' \times K \times 3$. K is the number of points in the neighborhood of the

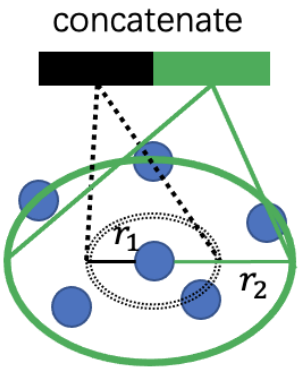

Fig. 3.15. The example of the multi-scale grouping. Two radius, i.e., r_1 and r_2, are utilized.

centroid points, and each group corresponds to a small local area of point cloud. Although K changes between groups, the following PointNet layer can transform a variable number of points into a fixed length local area feature vector.

Ball query is utilized to find K neighboring points of each centroid point. In the ball query, if the distant between a point x_k and a centroid point x_c is smaller than a radius, the point and the centroid point are the same group, i.e., $X_g := \{x_k\}_{distant(x_k,x_c)\leq r}$, r is the radius. As we described before, the number of points in a group is fixed, it is K. If the number of points of group is larger than K, the K nearest points in X_g are selected. If the number of points of group is smaller than K, the group are filled by copying the centroid point.

PointNet layer Each group are inputed to a PointNet layer to calculate the features of the group. The PointNet layer contains a MLP and a MaxPool layer.

According to these three processes, the output of PointNet++ layer has two parts. The first parts is the subset of row point cloud, i.e., $X_s = \{x_{ij}\}_{j=1}^{m}$. The second part is the representation of the local of the point in the X_s, i.e., $F_s = \{f_{ij}\}_{j=1}^{m}$. Therefore, our low-resolution point cloud is the concatenation of the X_s and the F_s, i.e., low-resolution point cloud := $\text{cat}(X_s, F_s)$. $\text{cat}(\cdot)$ is the concatenation operation.

In addition, in our PointNet++ layer, we utilize multi-scale grouping, which is introduced in PointNet++ [65], too. Grouping layers with different scales, i.e., different radius, followed by according PointNets are applied to point cloud. A multi-scale feature is created by concatenating features from different scales. The multi-scale grouping is shown in Fig.3.15.

Grouped SAGConv module

Fig. 3.16 depicts the structure of the grouped SAGConv module (G-SAGConv in Fig. 3.12). A set of F-dimensional features from the layer before are input into this module.

The input of the module is $\mathcal{H} = \{h_j\}_{j=1}^{M}$, where M stands for the number of features in the input. For the first module, $\mathcal{H}$ is simply $\mathcal{X}$ and $F = 3$, which represents x, y, and z coordinates of the points.

The k-NN algorithm is used to build the graph for $\mathcal{H}$. We create numerous graphs with various $k = \{11, 41\}$ values to reduce the impact of k in the k-NN. A group of SAGConv inspired by ResNeXt [69] is used to process numerous graph signals simultaneously. The fundamental principle of ResNeXt is that the input is first divided into branches, each of which processes the input on its own before fusing the results.

Dynamic graph convolution, which permits the graph topology to change at each layer, outperforms a fixed graph structure, according to [58, 70]. As a result, distinct graphs are built for different G-SAGConv modules.

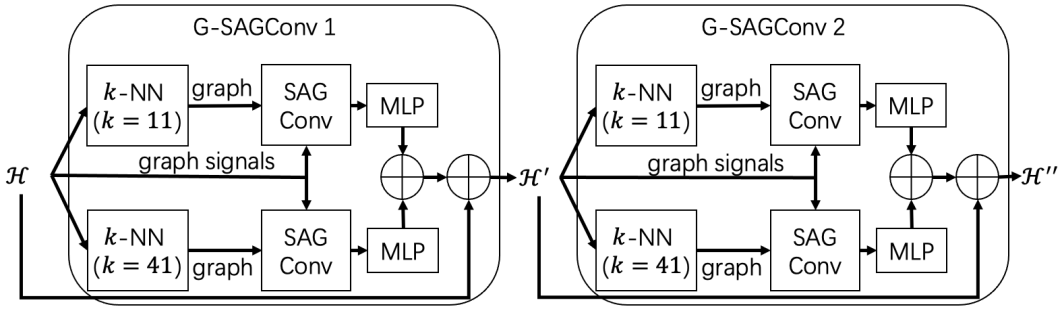

Fig. 3.16. Structure of the grouped SAGConv module. We have two modules (G-SAGConv 1 and G-SAGConv 2), and two k-NNs are applied to the input to create dynamic edges in each module.

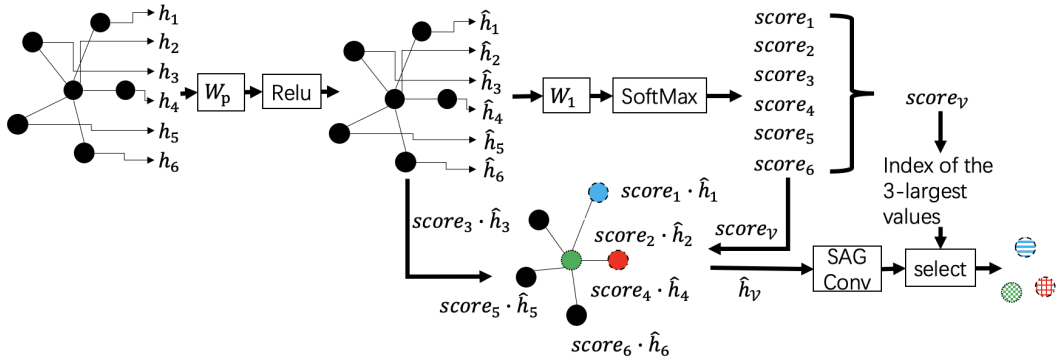

Fig. 3.17. Example of our graph pooling for $w = 3$.

Graph Pooling

Effective graph pooling algorithms are a popular topic in GNNs and graph signal processing [53, 54], as we discussed in chapter 2. Early investigations made use of global node representation pooling and graph coarsening methods.

In our method, we propose a score-based graph pooling method inspired by GraphU-net. Fig. 3.17 depicts our graph pooling. First, we embed node-wise features as follows:

$$\hat{h}_v = \sigma(W_{\mathrm{p}} \cdot h_v), v \in V, \tag{3.21}$$

where W_{p} is the shared learnable weight. We then learn a score on v as follows:

$$\begin{aligned} \mathrm{score}_v &= \mathrm{SoftMax}(W_1 \cdot \hat{h}_v) \\ &:= \frac{\exp\left(W_1 \cdot \hat{h}_v\right)}{\sum_{u \in V} \exp\left(W_1 \cdot \hat{h}_u\right)}, v \in V, \end{aligned} \tag{3.22}$$

where W_1 is the shared learnable weight. Subsequently, features on nodes are updated using the scores, i.e., $\hat{h}_V = \{\mathrm{score}_v \cdot \hat{h}_v\}$.

Subsequently, we update each node-wise features of a node using the SAGConv as follows:

$$\hat{h}'_V = \mathrm{SAGConv}(\hat{h}_V), \tag{3.23}$$

Finally, we arrange the nodes in descending order according to their scores and select the top w nodes as follows:

$$h_{\mathrm{select}} = \hat{h}'_V[\mathrm{idx}_{\mathrm{select}}] \tag{3.24}$$

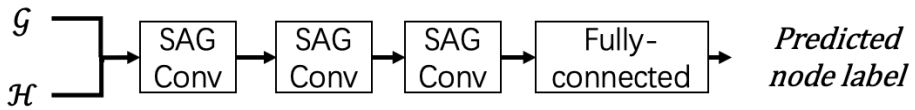

Fig. 3.18. Architecture of the node classification network (i.e., forward propagation), where $\mathcal{G}$ is the graph and $\mathcal{H}$ is the node-wise features.

where h_{select} is the set of node-wise features in the smaller graph, $\text{idx}_{\text{select}} = \text{rank}\left(\{\text{score}_v\}_V, w\right)$ is the indices of the w nodes selected, in which $\text{rank}\left(\cdot\right)$ is the node ranking operation that returns indices of the w highest scores.

We choose w nodes from the learnt scores, as shown in Fig. 3.17. We also use the SAGConv to process the features of the chosen nodes, in contrast to GraphU-net, which outputs the selected w nodes directly.

Hierarchical Prediction Architecture

We use the intermediate supervision technique [71] and develop a hierarchical prediction architecture to fully leverage the hierarchical characteristics (Fig.3.12). In conclusion, we take into account the loss at each G-SAGConv module while calculating the overall loss function.

There are two G-SAGConv modules for each phase. Max and average pooling layers were added to it. These outputs are then combined and used as the input for a fully connected layer. We calculate the prediction label and the classification loss for each stage. The overall classification loss and final prediction labels are the summation of the losses of each phase and the prediction labels of each phase. The following is a representation of this processing:

$$prediction = \sum_{i=1}^{P} prediction_i, \tag{3.25}$$

$$loss = \sum_{i=1}^{P} loss_i, \tag{3.26}$$

where $prediction_i$ is the final prediction value of the ith phase, $loss_i$ is the cross-entropy loss of the ith phase. $prediction$ and $loss$ are the overall prediction value and classification loss, respectively, and P is the number of phases.

The benefit of this architecture is that by combining the outcomes of the several phases, we can produce predictions that are more dependable and robust.

3.4.2 Node Classification

Fig.3.18 depicts the overall architecture of the node classification network based on the SAGConv. To predict the label for each node, we simply concatenate three SAGConv layers and a fully connected layer. The cross-entropy loss is used as the loss function.

3.5 Experimental Results

In this section, we describe the details of the dataset and experimental results for point cloud and node classification experiments.

3.5.1 Point Cloud Classification

Dataset

We utilize the ModelNet dataset [66] for the classification of point clouds. A total of 12,308 computer-aided design (CAD) models in 40 categories are available on ModelNet40, including 2,468 for testing and 9,840 for training. 4,899 CAD models total, 3,991 for training and 908 for testing, are included in ModelNet10. We used the same data structure as earlier 3D point cloud classification methods: 1,024 points were evenly sampled among the faces of the CAD mesh. All point clouds were initially adjusted to be contained within a unit sphere.

Settings and Evaluation Metric

The settings of hyper parameters of the point cloud classification network are shown in Table 3.1.

Table 3.1: The hyper parameters of the point cloud classification network. k is the value of k-NN, w is the number of selected nodes. For Pointnet++ layer, S is the number of sampled points, r is the radius of each group, D is the number of points of each group.

Learning rate	0.001335			
Drop out	0.3			
Graph pooling layername	k	w	[input channels, output channels]	
Graph pooling 0	41	512	[64,64]	
Graph pooling 1	41	128	[128,128]	
Graph pooling 2	41	128	[128,128]	
Graph convolution layername	k	[input channels, output channels]		
G-SAGConv 0	[11,41]	[3,64]		
G-SAGConv 1	[11,41]	[64,64]		
G-SAGConv 2	[11,41]	[64,64]		
G-SAGConv 3	[11,41]	[64,128]		
G-SAGConv 4	[11,41]	[128,256]		
G-SAGConv 5	[11,41]	[256,256]		
G-SAGConv 6	[11,41]	[128,128]		
G-SAGConv 7	[11,41]	[128,128]		
G-SAGConv 8	[11,41]	[128,256]		
G-SAGConv 9	[11,41]	[256,256]		
Pointnet++ layer name	S	r	D	[input channels, output channels]
Pointnet++	512	0.2	32	[3,64]
		0.4	128	[3,64]

We evaluated the accuracy of classification with other networks created specifically for point cloud classification. Owing to the class imbalance in the dataset, we used two performance metrics to assess the results: Average accuracy of all test instances (OA) and average accuracy of all shape classes (mAcc).

Results and Discussion

Table 4.2 provides a summary of the results, and the related references were used to determine the scores for the alternative methods. We discovered that our network had higher accuracy than the competing methods on both OA and mAcc.

We compare graph-based methods, including ours, below. The methods in [72–76] are spectral methods, and the methods in [58, 70, 77–81] are spatial methods.

The graph convolution used by DPAM [72] is GCN [19], and only the node-wise features of the one-hop neighboring nodes are utilized. The graph convolution process of RGCNN [73], 3DTI-Net [76], PointGCN [75], and LocalSpaceGCN [74] is carried out in the graph Fourier domain. They would overlook the local spatial information while designing the global spectral response. In addition, our technique can extract the structuralfeatures of one-hop neighbors in a single graph convolution operation, in contrast to the other spatial methods [58,70,77–81]. These might be plausible explanations for why our method performs better than other graph-based approaches.

Table 3.2: Comparison results of the 3D shape classification on the ModelNet benchmark. OA indicates the average accuracy of all test instances, and mAcc indicates the average accuracy of all shape categories. The symbol "-" indicates that the results are not available from the references.

Type	Method		ModelNet40		ModelNet10	
			OA	mAcc	OA	mAcc
Pointwise MLP Methods	PointNet [68]		89.2%	86.2%	-	-
	PointNet++ [65]		90.7%	-	-	-
	SRN-PointNet++ [82]		91.5%	-	-	-
	PointASNL [83]		93.2%	-	95.9%	-
Convolution-based Methods	PointConv [84]		92.5%	-	-	-
	A-CNN [85]		92.6%	90.3%	95.5%	95.3%
	SFCNN [86]		92.3%	-	-	-
	InterpCNN [87]		93.0%	-	-	-
	ConvPoint [88]		91.8%	88.5%	-	-
Graph-based Methods	Spectral Methods	DPAM [72]	91.9%	89.9%	94.6%	94.3%
		RGCNN [73]	90.5%	87.3%	-	-
		3DTI-Net [76]	91.7%	-	-	-
		PointGCN [75]	89.5%	86.1%	91.9%	91.6%
		LocalSpecGCN [74]	92.1%	-	-	-
	Spatial Methods	ECC [58]	87.4%	83.2%	90.8%	90.0%
		KCNet [77]	91.0%	-	94.4%	-
		DGCNN [70]	92.2%	90.2%	-	-
		LDGCNN [78]	92.9%	90.3%	-	-
		Hassani et al. [79]	89.1%	-	-	-
		ClusterNet [80]	87.1%	-	-	-
		Grid-GCN [81]	93.1%	91.3%	97.5%	97.4%
		SAGConv	**93.5%**	**91.3%**	**98.3%**	**97.7%**

3.5.2 Effect of structural features

We further confirm the effect of each structural features in this section.

Base line

Our base line is GraphSAGE [29]. As we described before, the GraphSAGE just aggregate the node-wise features of neighboring nodes, then integrate the node-wise features of the target node. It can be summarized as follows:

$$\begin{aligned} h_{N(v)} &= \frac{1}{|N(v)|} \sum_{u \in N(v)} \text{cat}(h_u), \\ h'_v &= \sigma\left(W \cdot \text{cat}\left(h_v, h_{N(v)}\right)\right), \end{aligned} \tag{3.27}$$

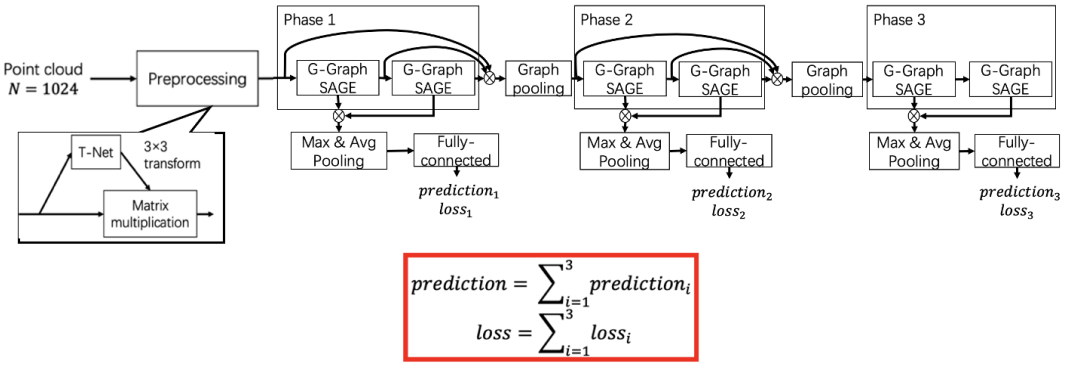

Fig. 3.19. Architecture of the sampled point cloud classification network, which utilize GraphSAGE as graph convolution operation.

Table 3.3: The settings of the simplified point cloud classification network using GraphSAGE as graph convolution operations. k is the value of k-NN, w is the number of selected nodes.

Learning rate	0.00135		
Drop out	0.3		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
GraphSAGE 0	[11,15]	[3,64]	
GraphSAGE 1	[11,15]	[64,64]	
GraphSAGE 2	[11,15]	[64,64]	
GraphSAGE 3	[11,15]	[64,128]	
GraphSAGE 4	[11,15]	[128,256]	
GraphSAGE 5	[11,15]	[256,256]	

where cat$(\cdot)$ denotes the concatenation operation, W is a learnable matrix, and h'_v is convolved features on v. We apply GraphSAGE into a sampled point cloud classification network, which we introduced in section 3.4. The architecture of the network is shown in Fig.3.19. Finally, we conduct experiments on point cloud classification. The graph pooling layer is introduced in section 3.4.

The settings are summarized in Table 3.3

Effect of feature angle and basis vector

To validate the effect of the feature angle, we introduce the feature angle into GraphSAGE. The graph convolution can be summarized as follows:

$$\begin{aligned} h_{N(v)} &= \frac{1}{|N(v)|} \sum_{u \in N(v)} \text{cat}(h_u, \text{fa}_{uv}), \\ h'_v &= \sigma\left(W \cdot \text{cat}\left(h_v, h_{N(v)}\right)\right), \end{aligned} \tag{3.28}$$

where cat$(\cdot)$ denotes the concatenation operation, W is a learnable matrix, and h'_v is convolved features on v. We name it as GraphSAGE-fa. We apply it into a sampled point cloud classification network, which we introduced in section 3.4. The architecture of the network is shown in Fig.3.20. Finally, we conduct experiments on point cloud classification. The graph pooling layer is introduced in section 3.4.

The settings are summarized in Table 3.4. In additional, we also confirm the effect of

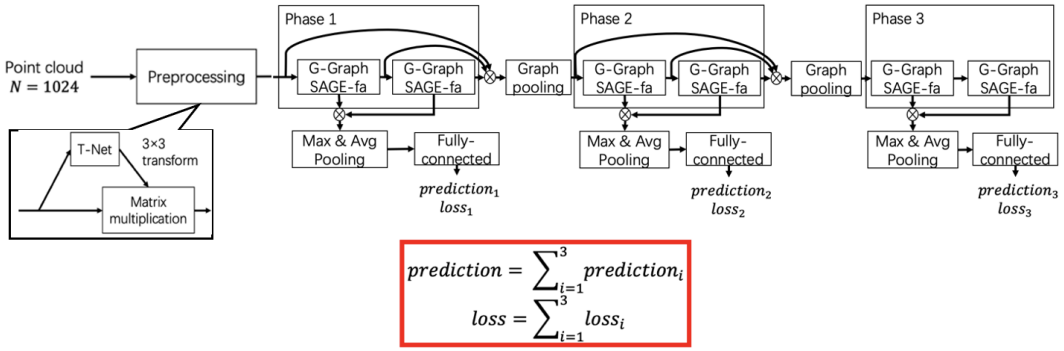

Fig. 3.20. Architecture of the sampled point cloud classification network, which utilize GraphSAGE-fa as graph convolution operation.

Table 3.4: The settings of the simplified point cloud classification network using GraphSAGE with feature angle as graph convolution operations. k is the value of k-NN, w is the number of selected nodes.

Learning rate	0.00135		
Drop out	0.3		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
GraphSAGE-fa 0	[11,15]	[3,64]	
GraphSAGE-fa 1	[11,15]	[64,64]	
GraphSAGE-fa 2	[11,15]	[64,64]	
GraphSAGE-fa 3	[11,15]	[64,128]	
GraphSAGE-fa 4	[11,15]	[128,256]	
GraphSAGE-fa 5	[11,15]	[256,256]	

different basis vector. We use nearest basis vector, max basis vector, average basis vector and learned basis vector, which are introduced in section 3.3, as the basis vector separately, when we calculate the feature angle.

Effect of feature distance

Based on the GraphSAGE, we introduce the feature distance into it. We named the graph convolution as GraphSAGE-fd. The GraphSAGE-fd can be summarized as follows:

$$\begin{aligned} h_{N(v)} &= \frac{1}{|N(v)|} \sum_{u \in N(v)} \mathrm{cat}(h_u, \mathrm{fd}_{uv}), \\ h'_v &= \sigma\left(W \cdot \mathrm{cat}\left(h_v, h_{N(v)}\right)\right), \end{aligned} \tag{3.29}$$

where $\mathrm{cat}(\cdot)$ denotes the concatenation operation, W is a learnable matrix, and h'_v is convolved features on v. We apply it into a sampled point cloud classification network, which we introduced in section 3.4. The architecture of the network is shown in Fig.3.21. Finally, we conduct experiments on point cloud classification. The graph pooling layer is introduced in section 3.4.

The settings are summarized in Table 3.5

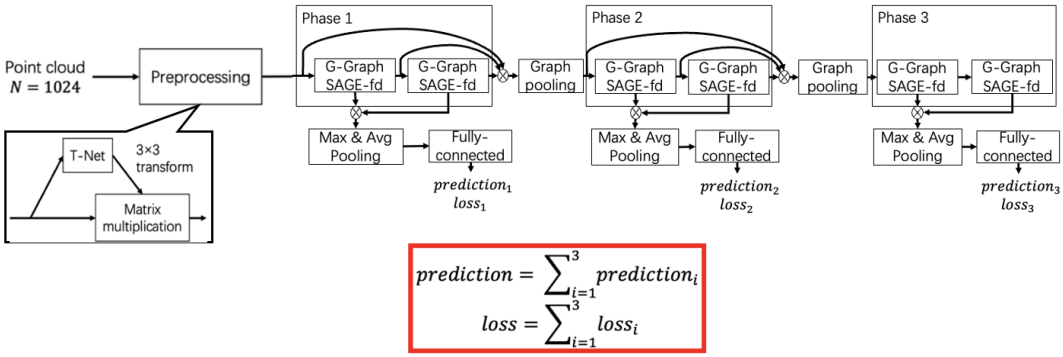

Fig. 3.21. Architecture of the sampled point cloud classification network, which utilize GraphSAGE-fd as graph convolution operation.

Table 3.5: The settings of the simplified point cloud classification network using GraphSAGE with feature distance as graph convolution operations. k is the value of k-NN, w is the number of selected nodes.

Learning rate	0.00135		
Drop out	0.3		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
GraphSAGE-fd 0	[11,15]	[3,64]	
GraphSAGE-fd 1	[11,15]	[64,64]	
GraphSAGE-fd 2	[11,15]	[64,64]	
GraphSAGE-fd 3	[11,15]	[64,128]	
GraphSAGE-fd 4	[11,15]	[128,256]	
GraphSAGE-fd 5	[11,15]	[256,256]	

Effect of relational embedding

To validate the effect of the realtional embedding, we introduce the relational embedding into GraphSAGE. The graph convolution can be summarized as follows:

$$\begin{aligned} h_{N(v)} &= \frac{1}{|N(v)|} \sum_{u \in N(v)} \text{cat}(h_u, \text{re}_{uv}), \\ h'_v &= \sigma\left(W \cdot \text{cat}\left(h_v, h_{N(v)}\right)\right), \end{aligned} \tag{3.30}$$

where $\text{cat}(\cdot)$ denotes the concatenation operation, W is a learnable matrix, and h'_v is convolved features on v. We name it as GraphSAGE-re. We apply it into a sampled point cloud classification network, which we introduced in section 3.4. The architecture of the network is shown in Fig.3.22. Finally, we conduct experiments on point cloud classification. The graph pooling layer is introduced in section 3.4.

The settings are summarized in Table 3.6.

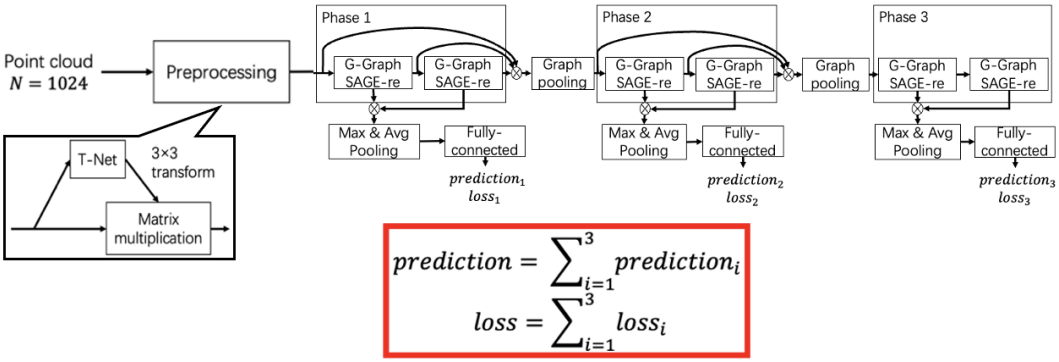

Fig. 3.22. Architecture of the sampled point cloud classification network, which utilize GraphSAGE-re as graph convolution operation.

Table 3.6: The settings of the simplified point cloud classification network using GraphSAGE with relational embedding as graph convolution operations. k is the value of k-NN, w is the number of selected nodes.

Learning rate	0.00135		
Drop out	0.3		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
GraphSAGE-re 0	[11,15]	[3,64]	
GraphSAGE-re 1	[11,15]	[64,64]	
GraphSAGE-re 2	[11,15]	[64,64]	
GraphSAGE-re 3	[11,15]	[64,128]	
GraphSAGE-re 4	[11,15]	[128,256]	
GraphSAGE-re 5	[11,15]	[256,256]	

Effect of the set of feature angle, feature distance, and relational embedding

Furthermore, we introduce the set of feature angle, feature distance, and relational embedding into GraphSAGE. It becomes our SAGConv. Our SAGConv is

$$\begin{aligned} h_{N(v)} &= \frac{1}{|N(v)|} \sum_{u \in N(v)} \mathrm{cat}(h_u, \mathrm{fa}_{uv}, \mathrm{fd}_{uv}, \mathrm{re}_{uv}), \\ h'_v &= \sigma\left(W \cdot \mathrm{cat}\left(h_v, h_{N(v)}\right)\right), \end{aligned} \tag{3.31}$$

where $\mathrm{cat}(\cdot)$ denotes the concatenation operation, W is a learnable matrix, and h'_v is convolved features on v. We apply it into a sampled point cloud classification network, which we introduced in section 3.4. The architecture of the network is shown in Fig.3.23. Finally, we conduct experiments on point cloud classification. The graph pooling layer is introduced in section 3.4.

The settings are summarized in Table 3.7

Results and analysis

First, we confirmed the effect of the different basis vectors. The results, i.e., the classification OA, are shown in Table 3.8. The feature angle utilizing the nearest basis vector does not contribute to the performance. Although the feature angle utilizing the learned basis vector

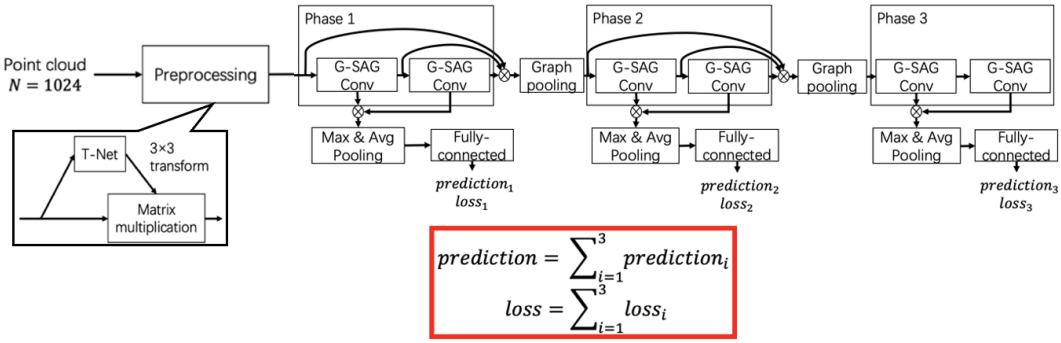

Fig. 3.23. Architecture of the sampled point cloud classification network, which utilize SAGConv as graph convolution operation.

Table 3.7: The settings of the simplified point cloud classification network using SAGConv. k is the value of k-NN, w is the number of selected nodes.

Learning rate	0.00135		
Drop out	0.3		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
SAGConv 0	[11,15]	[3,64]	
SAGConv 1	[11,15]	[64,64]	
SAGConv 2	[11,15]	[64,64]	
SAGConv 3	[11,15]	[64,128]	
SAGConv 4	[11,15]	[128,256]	
SAGConv 5	[11,15]	[256,256]	

has the best performance, the disparity with the feature angle utilizing the max basis vector is just 0.1% , and the gap with the feature angle using the average basis vector is 0.2% . Although, we also think the feature angle using learned basis vector can fit to more tasks.

Then, to confirm the effect of each structural features, we also conduct experiments using GraphSAGE-fa, GraphSAGE-fd, GraphSAGE-re and SAGConv for 3D point cloud classification. The classification OA is shown in Table 3.9 and Fig.3.24. The classification accuracy of GraphSAGE for point cloud is 90.0% . The accuracy of GraphSAGE-fa is 90.6% . The accuracy is raised by 0.6% using the feature angle (fa_{uv}). The accuracy of GraphSAGE-fd is 91.3% . The accuracy is raised by 1.3% using the feature distance (fd_{uv}). The accuracy of GraphSAGE-re is 91.4% . The accuracy is raised by 1.4% using the relational embedding (re_{uv}). The accuracy of SAGConv is 92.1% . The accuracy is raised by 2.1% , using the set of structural features.

Finally, to further confirm the percentage of contribution of each structural features, we construct a spatial GC using feature angle and feature distance together. We name it as GraphSAGE-fad. The GraphSAGE-fad is summarized as follows:

$$\begin{aligned} h_{N(v)} &= \frac{1}{|N(v)|} \sum_{u \in N(v)} \mathrm{cat}(h_u, \mathrm{fa}_{uv}, \mathrm{fd}_{uv}), \\ h'_v &= \sigma\left(W \cdot \mathrm{cat}\left(h_v, h_{N(v)}\right)\right), \end{aligned} \tag{3.32}$$

where $\mathrm{cat}(\cdot)$ denotes the concatenation operation, W is a learnable matrix, and h'_v is convolved features on v. We apply it into a sampled point cloud classification network, which

Table 3.8: Comparison results of the 3D shape classification on the ModelNet benchmark. OA indicates the average accuracy of all test instances.

	Method	ModelNet40
		OA
Different Basis Vector	GraphSAGE-fa (nearest basis vector)	90.0%
	GraphSAGE-fa (max basis vector)	90.4%
	GraphSAGE-fa (average basis vector)	90.3 %
	GraphSAGE-fa (learned basis vector)	**90.6 %**

Table 3.9: Comparison results of the 3D shape classification on the ModelNet benchmark. OA indicates the average accuracy of all test instances.

	Method	ModelNet40
		OA
Different Graph Convolution Layer	GraphSAGE	90.0%
	GraphSAGE-fa (learned basis vector)	90.6%
	GraphSAGE-fd	91.3 %
	GraphSAGE-re	91.4 %
	SAGConv	92.4%

we introduced in section 3.4. The architecture of the network is shown in Fig.3.25. Finally, we conduct experiments on point cloud classification. The graph pooling layer is introduced in section 3.4. The settings are summarized in Table 3.10

As we shown in Fig. 3.24, when compare the GraphSAGE with GraphSAGE-fa, we can see that the feature angle increased the accuracy by 0.6 %. When we compare the GraphSAGE-fa with GraphSAGE-fad, the accuracy is raised by 1.3% using the feature distance (fd_{uv}). When we compare the GraphSAGE-fad with SAGConv, the accuracy is raised by 0.2% using the relational embedding (re_{uv}). Therefore, the feature distance is the most effective structural feature in our structural features. The effectiveness of the feature angle is second, and the effectiveness of relational embedding is third. These results provide the evidence that each structural feature is effective.

3.5.3 Effect of graph pooling

Here, we conduct experiments on 3D point clouds classification task for ModelNet 40 [66], to confirm the effect of our graph pooling layer. We use three sampled 3D point clouds classification networks. Our baseline is Graph U-net [60], i.e., gPool.

The structure of the first network has two layers of gPool. It is shown in Fig. 3.26 and the settings are shown in Table 3.11.

The structure of the second network does not have graph pooling layer. It is shown in Fig. 3.27 and the settings are shown in Table 3.12.

The structure of the third network has two graph pooling layers, which we proposed in section 3.4.1. It is shown in Fig. 3.28 and the settings are shown in Table 3.13.

The classification results are summarized in Table 3.14. According to the results, we can know that our graph pooling layers are effective. Due to the graph pooling layers can select t important nodes, the network can ignore useless information. Therefore, our graph

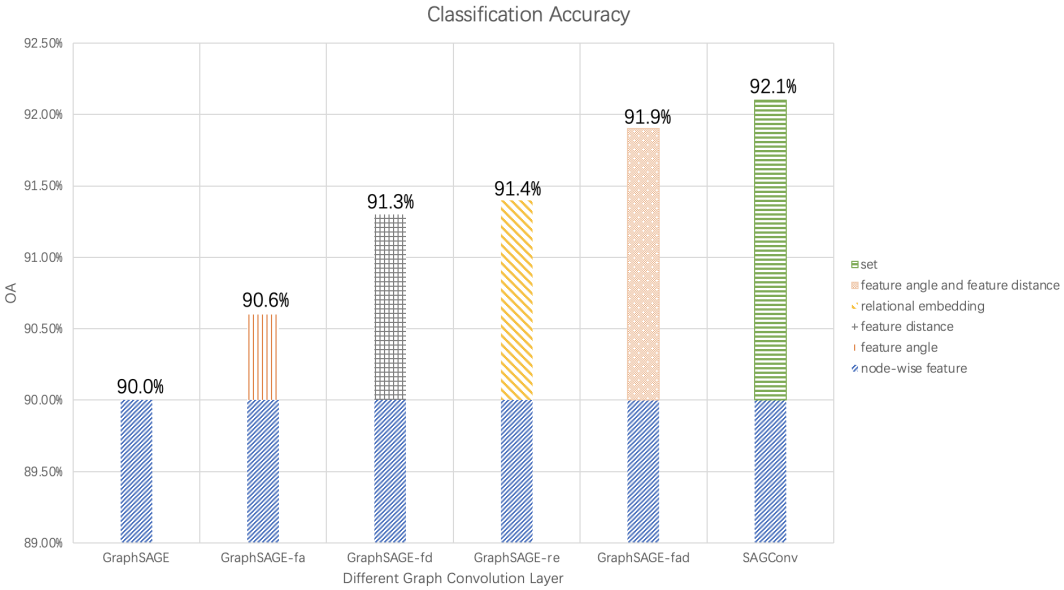

Fig. 3.24. The effect of each structural feature.

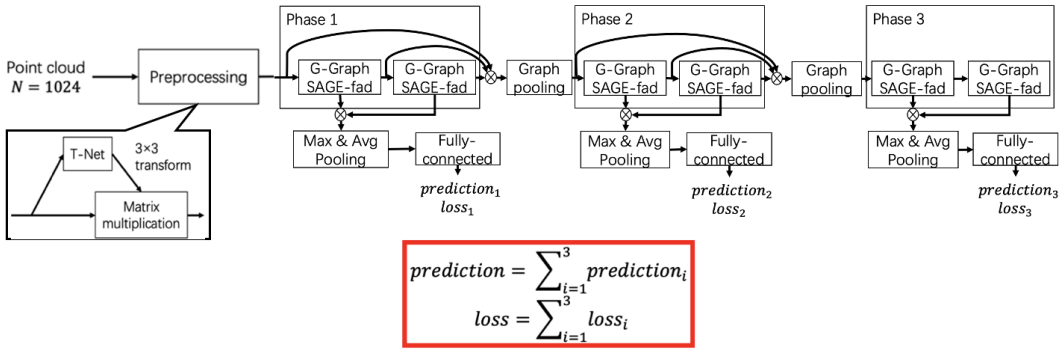

Fig. 3.25. Architecture of the sampled point cloud classification network, which utilize GraphSAGE-fad as graph convolution operation.

pooling layers can improve the performance of the networks. It also can reduce the need of the compute source.

3.5.4 Node Classification

To further validate the performance of SAGConv, we also conduct experiments on a node classification problem for Cora [67], a standard citation network dataset. In Cora, nodes represent papers, and edges indicate citations. It contains 2,708 nodes and 5,429 edges, and the dimension of each feature is 1,433. There are seven categories of node labels. The hyper parameters of the node classification network are summarized in Table 3.15.

The classification results with representative methods are shown in Table 3.16. It is observed that our network also achieves a good classification accuracy on the citation networks while the network structure is rather simple.

Table 3.10: The settings of the simplified point cloud classification network using GraphSAGE-fad. k is the value of k-NN, w is the number of selected nodes.

Learning rate	0.00135		
Drop out	0.3		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
GraphSAGE-fad 0	[11,15]	[3,64]	
GraphSAGE-fad 1	[11,15]	[64,64]	
GraphSAGE-fad 2	[11,15]	[64,64]	
GraphSAGE-fad 3	[11,15]	[64,128]	
GraphSAGE-fad 4	[11,15]	[128,256]	
GraphSAGE-fad 5	[11,15]	[256,256]	

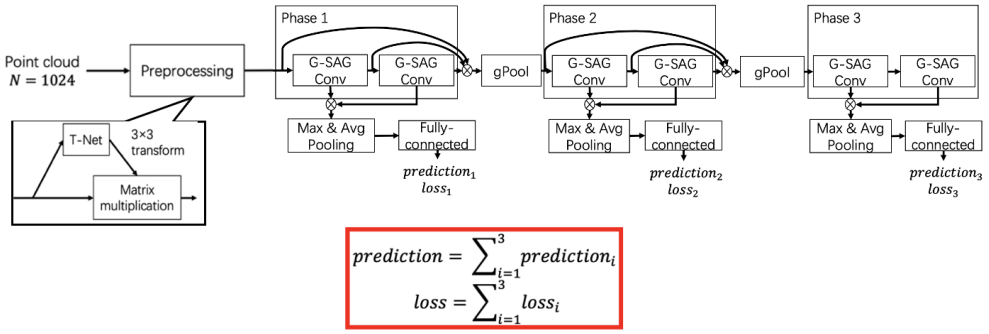

Fig. 3.26. The networks are utilized to confirm the effect of our graph pooling layer. It has two gPool layers.

3.6 Conclusion

Here, we suggest SAGConv, a new method to spatial graph convolution. It uses feature distance, feature angle, and relationship embedding to represent the neighborhood in detail in feature space. We also suggest SAGConv-based networks for classifying nodes and graphs. Our method outperforms other methods in experiments.

Table 3.11: The settings of the simplified point cloud classification network with two gPool layers (Fig. 3.26). w is the number of selected nodes.

Learning rate	0.00135	
Drop out	0.3	
Graph pooling layername	w	[input channels, output channels]
gPool 0	512	[64,64]
gPool 1	128	[128,128]
Graph convolution layername	k	[input channels, output channels]
SAGConv 0	[11,15]	[3,64]
SAGConv 1	[11,15]	[64,64]
SAGConv 2	[11,15]	[64,64]
SAGConv 3	[11,15]	[64,128]
SAGConv 4	[11,15]	[128,256]
SAGConv 5	[11,15]	[256,256]

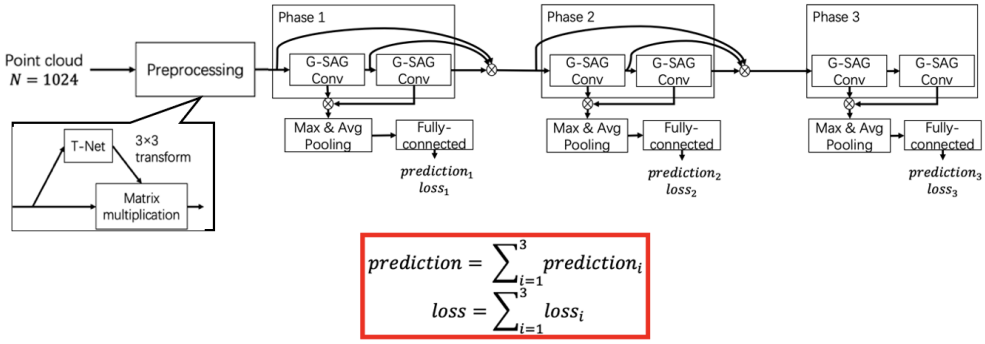

Fig. 3.27. The networks are utilized to confirm the effect of our graph pooling layer. It is the sampled network without graph pooling layers.

Table 3.12: The settings of the simplified point cloud classification network without graph pooling layers (Fig. 3.27). k is the value of k-NN.

Learning rate	0.00135	
Drop out	0.3	
Graph convolution layername	k	[input channels, output channels]
SAGConv 0	[11,15]	[3,64]
SAGConv 1	[11,15]	[64,64]
SAGConv 2	[11,15]	[64,64]
SAGConv 3	[11,15]	[64,128]
SAGConv 4	[11,15]	[128,256]
SAGConv 5	[11,15]	[256,256]

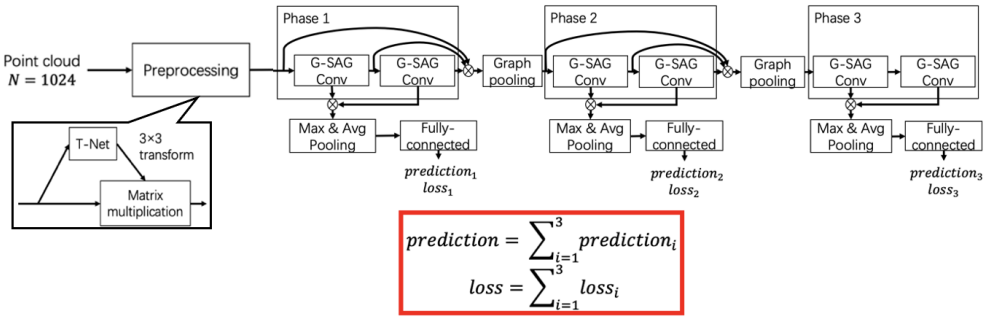

Fig. 3.28. The networks are utilized to confirm the effect of our graph pooling layer. It is the sampled networks with two graph pooling layers that is introduced in section 3.4.1.

Table 3.13: The settings of the simplified point cloud classification network with two graph pooling layers (Fig. 3.28). k is the value of k-NN, w is the number of selected nodes.

Learning rate	0.00135		
Drop out	0.3		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
SAGConv 0	[11,15]	[3,64]	
SAGConv 1	[11,15]	[64,64]	
SAGConv 2	[11,15]	[64,64]	
SAGConv 3	[11,15]	[64,128]	
SAGConv 4	[11,15]	[128,256]	
SAGConv 5	[11,15]	[256,256]	

Table 3.14: Comparison results of the 3D shape classification on the ModelNet benchmark. OA indicates the average accuracy of all test instances.

	Method	ModelNet40
		OA
Different networks	SAGConv + gPool (baseline)	91.2%
	SAGConv (without graph pooling layers)	91.1%
	SAGConv (with two graph pooling layers)	**92.4%**

Table 3.15: The hyper parameters of node classification network.

Learning rate	0.1	
Drop out	0.3	
layername	[input channels, output channels]	
	Cora	Pubmed
SAMGC 0	[1433,128]	[500,128]
SAMGC 1	[128,256]	[128,256]
SAMGC 2	[256,1024]	[256,1024]

Table 3.16: Node classification results for the Cora benchmark. The symbol "-" indicates that the results are not available from the references.

Method	Accuracy Cora
CayleyNet [33]	81.2%
GCN [19]	81.4%
GraphSAGE [29]	82.1%
GAT [30]	83.0%
SAGConv	**83.0%**

Chapter 4

Structure-Aware Multi-Hop Graph Convolution for Graph Neural Networks

4.1 Introduction

Due to large-scale datasets and improvements in processing power, deep neural networks have, as we previously noted, achieved exceptional success in identifying, segmenting, and recognizing regularly structured material such as photos and videos [15–17]. It is difficult to process multiple irregular data sets without a predetermined sample order in the actual world using conventional deep learning techniques. Examples of this include 3D point clouds, social network opinions, and traffic network passenger counts. Data with irregular structures could be processed more efficiently by being converted to graph-structured data. Graph neural networks (GNNs) have gotten a lot of interest in this regard [25, 44, 45].

In existing GNNs, graph convolutions (GCs), often referred to as graph filters, are frequently utilized as the time and space-domain equivalent of classic convolution. The main idea of GC algorithms is to iteratively combine the features of neighboring node before integrating them with the features of target node [25, 26, 63]. Existing techniques for a single step of GCs mainly use node-wise features of the one-hop neighborhood [19, 29, 30]. They occur over-smoothing [25, 28]. Over-smoothing makes node-wise features of nodes to be the same where edge exists. Therefore, over-smoothing may influences the performance of GNNs.

One of the strategies to resolve the over-smoothing is to improve the expressive of GCs. Therefore, we see two limitations of expressive as being present:

1. Particularly in the feature space, the *structural information* of the surrounding neighboring nodes may be lost using the existing methods.

2. Even while the multi-hop neighborhood might have some helpful information, the single step GC cannot make use of this.

In other words, if we can effectively use 1) comprehensive structure information of the nearby neighboring nodes in the feature space, and 2) GCs that take into account multi-hop neighborhoods, the expressive of GCs can be improved. Furthermore, the performance of GNNs may also be improved.

In this chapter, we propose a new GC to overcome the two limitations mentioned before. Our proposed GC uses two methods.

First, we utilize three structural features to characterize the surrounding structure in the feature space: feature angle, feature distance, and relational embedding, which we proposed in 3. We concatenate them with node-wise features. In this chapter, we also significantly extend the spatial graph convolution by proposing a neighbor-wise learnable average aggregation.

The second is the suggestion of a GC aggregating multi-hop characteristics. In our method, multi-hop features are aggregated concurrently in the one-step GC as opposed to the iteration of one-hop feature aggregation.

The two methods can be simultaneously used in tandem to offer our structure-aware multi-hop GC (SAMGC). The following is a summary of our contributions:

1. In the feature space, we may extract precise structural information about surrounding neighboring nodes, and the multi-hop convolution gathers useful data about indirect neighborhoods. It enables the convolved nodes to contain richer information than is currently possible with existing approaches.

2. Based on the SAMGC, we construct two GNNs for graph and node classification tasks. 3D point clouds are used as a sample example for graph classification. A layer of PointNet++ [65] and a layer of graph pooling are mixed with SAMGC layers. The ModelNet [66] dataset experimentations shown that our method has higher classification accuracy than existing methods.

   For node classification, we cascade three SAMGC layers with a fully connected layer. We showed how our method may be compared to other methods that are explicitly designed for node classification in the Cora dataset [67] and the Pubmed dataset [89] datasets.

Notation: An undirected graph is defined as $G = (V, E)$ where V is a set of nodes, and E is a set of edges. The adjacency matrix of G is denoted as A. $\widetilde{D}$ is the diagonal degree matrix.

Here, $h_v := [\mathrm{h}_{v1}, \ldots, \mathrm{h}_{vj}, \ldots, \mathrm{h}_{vC}]^\top \in \mathbb{R}^C$ represents a feature vector on the node $v \in V$, and C is the number of features in h_v.

The non-linearity function is denoted as $\sigma(\cdot)$. The number of neighborhood hops is t. The set of i-hop neighboring nodes is $N_i(\cdot)$ in which its cardinality is denoted as $|N_i(\cdot)|$. A multilayer perceptron (MLP) layer is represented as $\mathrm{MLP}(\cdot)$. A channel-wise max-pooling is denoted as $\mathrm{MaxPool}(\cdot)$. The vector concatenation operation is denoted as $\mathrm{cat}(\cdot)$.

4.2 Preliminaries

The fundamental idea behind GCs is to incrementally aggregate the data of one-hop neughbor nodes and then integrate the aggregated data with that of the target node [25]. The two types of GC methods now in use are spectral and spatial methods.

4.2.1 Spectral methods

A GC operation is carried out for spectral methods in the graph Fourier domain. The graph Fourier basis is the name given to the eigenvector matrix of the graph Laplacian [15, 31]. Polynomial approximations are typically employed in place of eigendecomposition since it requires high processing complexity for big graphs [18, 32, 33].

In order to further streamline the method, a linear graph filter is suggested in [19]. It is represented as follows:

$$H_{GCN} = \sigma(\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}HW), \tag{4.1}$$

where $H := \{h_v\}_{v \in \mathcal{V}}$ is the set of node-wise features, $\widetilde{A} = A + I_n$ is the adjacency matrix with self-loops, and W is the trainable weight matrices. This assumes that the spectral filter is a linear function of the graph frequency, i.e., the eigenvalue λ.

4.2.2 Spatial methods

Spatial GC techniques immediately aggregate the node-wise features of one-hop neighboring nodes into the target node. A spatial GC is formed by aggregation and integration operations. It can be represented as follows:

$$h'_v = \text{integration}(h_{N_1(v)}, h_v), v \in \mathcal{V} \tag{4.2}$$

where

$$h_{N_1(v)} = \text{aggregation}(h_u), u \in N_1(v) \tag{4.3}$$

in which h'_v is the updated node-wise feature of the target node v. Furthermore, $h_{N_1(v)}$ is the aggregated node-wise feature at $N_1(v)$. The aggregation operation gathers and aggregates the features of one-hop neighbors. Aggregation by average or sum is frequently employed. Finally, during the integration operation, features are incorporated into the target node.

With an average aggregation, each one-hop neighbor is handled identically in the spatial GC method GraphSAGE [29]. Later, a graph attention network [30] based on the self-attention mechanism was developed to discriminate the importance of one-hop neighbors. However, as mentioned earlier, existing methods are limited to utilizing the node-wise features on $N_1(v)$.

Unlike previous methods, to improve the expressive of GCs, we first define structural features that describe the structure of one-hop neighboring nodes in the feature space and then install them into a spatial GC. Second, we use the information on multi-hop neighboring nodes. We use both of these two methods simultaneously for our new spatial GC, SAMGC.

4.3 SAMGC

The SAMGC is introduced in this section. In Fig.4.1, it is shown. As was already said, our goal is to make use of the structural information of the one-hop neighboring nodes in the feature space. During a spatial GC, we also use information from multi-hop neighbor nodes. The SAMGC includes these four components to accomplish this:

A. Structural Features;

B. One-hop Neighbor-wise Learnable Average Aggregation;

C. Multi-hop Neighborhood Aggregation;

D. Integration.

Its details are shown in Algorithm 2, and we sequentially introduce the four components.

4.3.1 Structural Features

We utilize three structural features in the feature space, which we introduce in chapter 3, to obtain precise structural information of the surrounding one-hop neighboring nodes:

- Feature angle;
- Feature distance;
- Relational embedding.

Below, we review these three features.

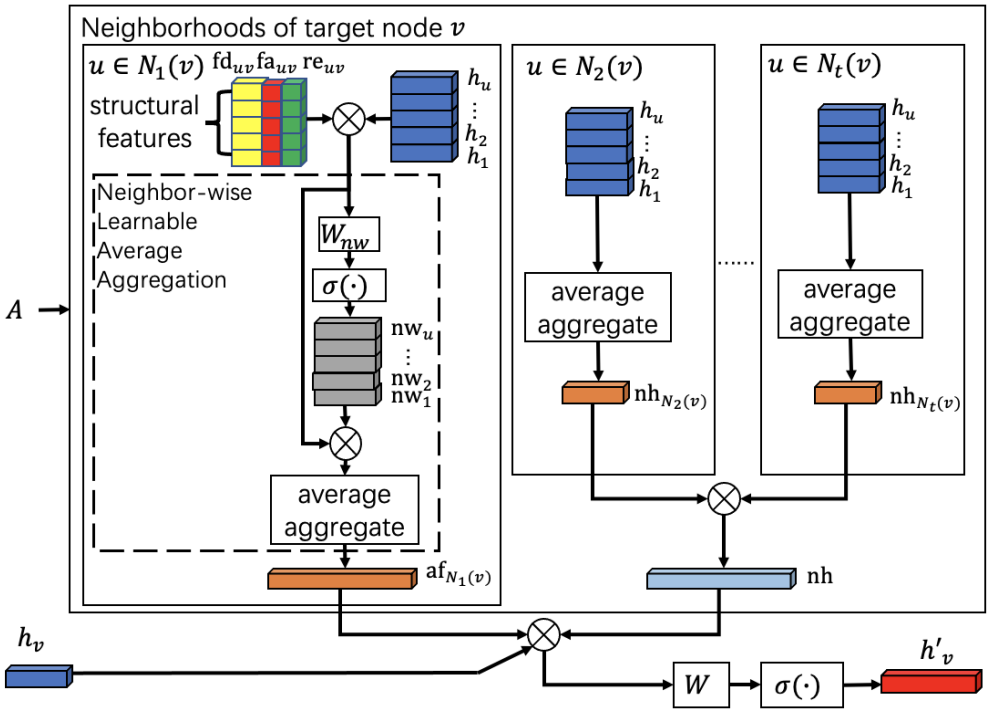

Fig. 4.1. SAMGC. $\otimes$ is the concatenation operation. W_{nw} and W are the learnable weights.

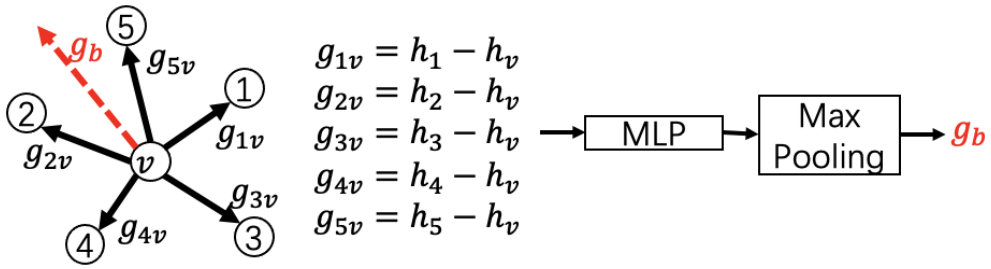

Fig. 4.2. Learning a base vector g_b. The numbers in the black circle are the node indices.

Feature angle

In order to describe the structure of the nearby one-hop neighborhoods, we first define *feature angles*. We create a collection of vectors pointing from v to the nodes that are within one hop of it in the neighborhood: $\mathcal{F}_{N_1(v)} = \{g_{uv} := h_u - h_v\}_{u \in N_1(v)}$. Subsequently, a basis vector g_b from $\mathcal{F}_{N_1(v)}$ is learned as follows:

$$g_b = \text{MaxPool}\left(\{\sigma\left(\text{MLP}\left(g_{uv}\right)\right)\}_{g_{uv} \in \mathcal{F}_{N_1(v)}}\right) \tag{4.4}$$

It is shown in Fig.4.2. Finally, the cosine of the angle between g_{uv} and g_b is computed as follows:

$$\text{fa}_{uv} = \cos(\theta_u) = \frac{g_{uv} \cdot g_b^{\text{T}}}{\|g_{uv}\| \cdot \|g_b\|}, \ g_{uv} \in \mathcal{F}_{N_1(v)} \tag{4.5}$$

The example is depicted in Fig. 4.3 (a).

Feature distance

The absolute difference between the elements of h_u and h_v represents the *feature distance* in feature space between one-hop neighbor nodes u and target node v. It has the following

Algorithm 2 SAMGC spatial graph convolution (i.e., forward propagation) algorithm

Input: graph $G = (V, E)$; input features $\{h_v, v \in V\}$; weight matrices W and W_{nw}

Output: Convolved features z_v for all $v \in V$

for $v \in \mathcal{V}$ **do**

$\mathcal{F}_{N_1(v)} = \{g_{uv} := h_u - h_v\}$

$g_b = \text{MaxPool}(\{\sigma(\text{MLP}(g_{uv}))\})$

$\text{fa}_{uv} = \cos(\theta_u) = \dfrac{g_{uv} \cdot g_b^{\mathrm{T}}}{\|g_{uv}\| \cdot \|g_b\|}, g_{uv} \in \mathcal{F}_{N_1(v)}$

$\text{fd}_{uv} = [|\text{h}_{u1} - \text{h}_{v1}|, ..., |\text{h}_{uC} - \text{h}_{vC}|]^\top$

$\text{re}_{uv} = \sigma(\text{MLP}(h_u - h_v)), u \in N_1(v)$

$\text{nw}_u = \sigma(W_{nw} \cdot \text{cat}(h_u, \text{fa}_{uv}, \text{fd}_{uv}, \text{re}_{uv})), u \in N_1(v)$

$\text{af}_{N_1(v)} = \frac{1}{|N_1(v)|} \sum_{u \in N_1(v)} \text{cat}(h_u, \text{fa}_{uv}, \text{fd}_{uv}, \text{re}_{uv}, \text{nw}_u)$

$\text{nh}_{N_2(v)} = \frac{1}{|N_2(v)|} \sum_{u \in N_2(v)} h_u$

$\text{nh}_{N_3(v)} = \frac{1}{|N_3(v)|} \sum_{u \in N_3(v)} h_u$

...

$\text{nh}_{N_t(v)} = \frac{1}{|N_t(v)|} \sum_{u \in N_t(v)} h_u$

$\text{nh} = \text{cat}\left(\text{nh}_{N_2(v)}, \text{nh}_{N_3(v)}, ..., \text{nh}_{N_t(v)}\right)$

$h'_v = \sigma\left(W \cdot \text{cat}\left(h_v, \text{af}_{N_1(v)}, \text{nh}\right)\right)$

end for

$z_v = h'_v, \forall v \in V$

return z_v

representation:

$$\text{fd}_{uv} = [|\text{h}_{u1} - \text{h}_{v1}|, ..., |\text{h}_{uC} - \text{h}_{vC}|]^\top. \tag{4.6}$$

The example is shown in Fig. 4.3 (b).

Relational embedding

Relational embedding illustrates the magnitude of the influence of one-hop neighbor nodes u on the target node v. We can learn it from the following from the distinction between h_v and h_u:

$$\text{re}_{uv} = \sigma(\text{MLP}\left(h_u - h_v\right)),\ u \in N_1(v). \tag{4.7}$$

re_{uv} is depicted in Fig. 4.3 (c).

4.3.2 One-hop Neighbor-wise Learnable Average Aggregation

As was said earlier, average aggregation is frequently employed for GCs. However, the specific neighborhood information can be lost. We suggest a neighbor-wise learnable average aggregation to fully leverage one-hop neighborhood information during aggregation. It consists of two parts:

1) Neighbor-wise learning;

2) Average aggregation.

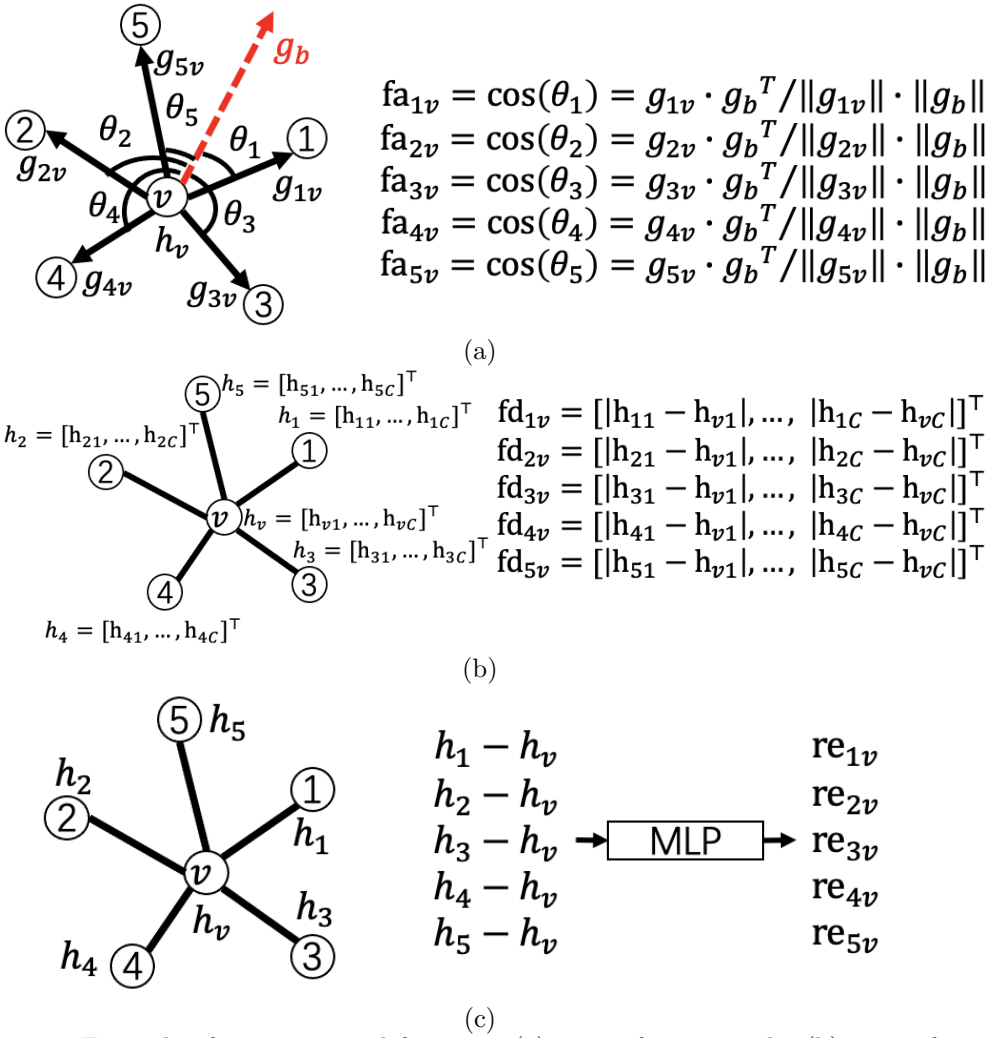

Fig. 4.3. Example of our structural features. (a) is our feature angle; (b) is our feature distance, h_{vj} is the element of h_v, C is the number of elements; (c) is our relational embedding.

Neighbor-wise learning

For each one-hop neighbor node, we integrate various features from section 4.3.1 and node-wise features using learnable weights W_{nw} in the manner described below, to fully use one-hop neighborhood features:

$$\mathrm{nw}_u = \sigma(W_{nw} \cdot \mathrm{cat}(h_u, \mathrm{fa}_{uv}, \mathrm{fd}_{uv}, \mathrm{re}_{uv})), u \in N_1(v). \tag{4.8}$$

This part is depicted in Fig.4.4.

Average Aggregation

After that, we compute general neighborhood features within one hop. Following is the concatenation and averaging of the node-wise features:

$$\mathrm{af}_{N_1(v)} = \frac{\sum_{u \in N_1(v)} \mathrm{cat}(h_u, \mathrm{fa}_{uv}, \mathrm{fd}_{uv}, \mathrm{re}_{uv}, \mathrm{nw}_u)}{|N_1(v)|}, \tag{4.9}$$

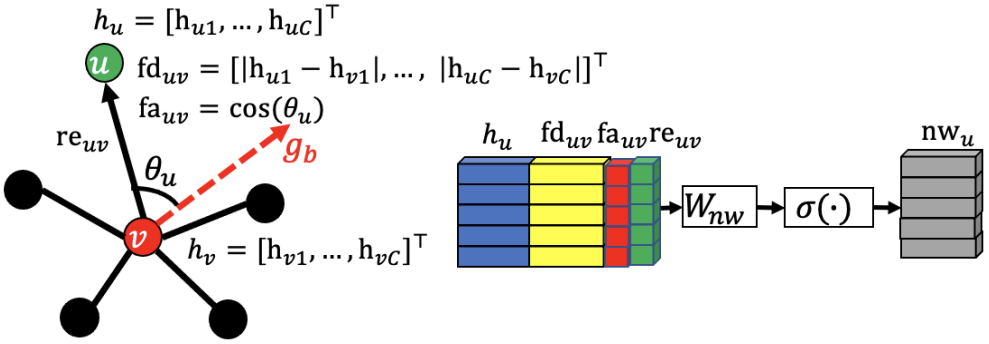

Fig. 4.4. Neighbor-wise learning. The node-wise feature of the 1st-hop neighboring node u is h_u and its corresponding structural features, fa_{uv}, fd_{uv}, and re_{uv}.

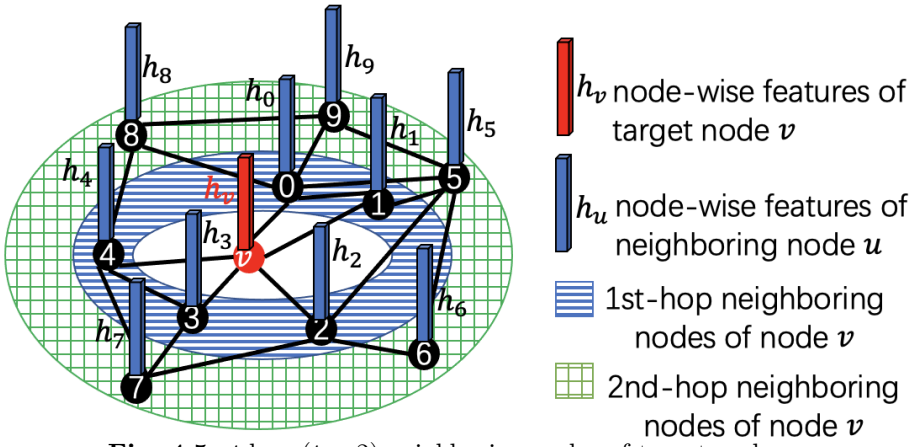

Fig. 4.5. t-hop ($t = 2$) neighboring nodes of target node v.

4.3.3 Multi-hop neighborhood aggregation

We leverage the features at t-hop ($t > 1$) surrounding neighbor nodes of the target node v to make use of the multi-hop neighboring information. It is shown in Fig.4.5.

We first introduce the node-wise feature aggregation of the ith-hop ($i \in [2, t]$) neighboring nodes. We compute matrix AN_i, which extracts the ith-hop neighborhood of v, using adjacency matrix A. The expression for the element in AN_i is as follows:

$$AN_{ivu} = \begin{cases} 1, & u \in N_i(v) \\ 0, & \text{otherwise} \end{cases} \tag{4.10}$$

We aggregate node-wise features on $N_i(v)$ as follows:

$$\text{nh}_{N_i(v)} = \frac{1}{|N_i(v)|}(AN_i \cdot H)_v, i \in [2, t]. \tag{4.11}$$

where $H := \{h_v\}_{v \in \mathcal{V}}$ is the set of node-wise features.

Finally, we concatenate $\text{nh}_{N_i(v)}$ as

$$\text{nh} = \text{cat}(\text{nh}_{N_2(v)}, \text{nh}_{N_3(v)}, \ldots, \text{nh}_{N_t(v)}). \tag{4.12}$$

4.3.4 Integration

The following integration of $\text{af}_{N_1(v)}$, nh, and the node-wise features on v serves as the final node-wise feature on v:

$$h'_v = \sigma(W \cdot \text{cat}(h_v, \text{af}_{N_1(v)}, \text{nh})), \tag{4.13}$$

where W is a learnable matrix. Hereafter, we denote the set of these operations as $h'_{\mathcal{V}} := \mathrm{SAMGC}(h_{\mathcal{V}})$.

4.4 GNN implementations

In this section, we introduce two GNNs using the SAMGC.

1. 3D point cloud classification network in Section 4.4.1;
2. Graph node classification network with cascaded SAMGCs in Section 4.4.2.

4.4.1 3D Point Cloud Classification

Fig. 4.6 depicts the overall structure of the SAMGC-based 3D point cloud classification network. Let us express the input point cloud as $\mathcal{X} = \{x_i\}_{i=1}^{N}$, where N is the number of points.

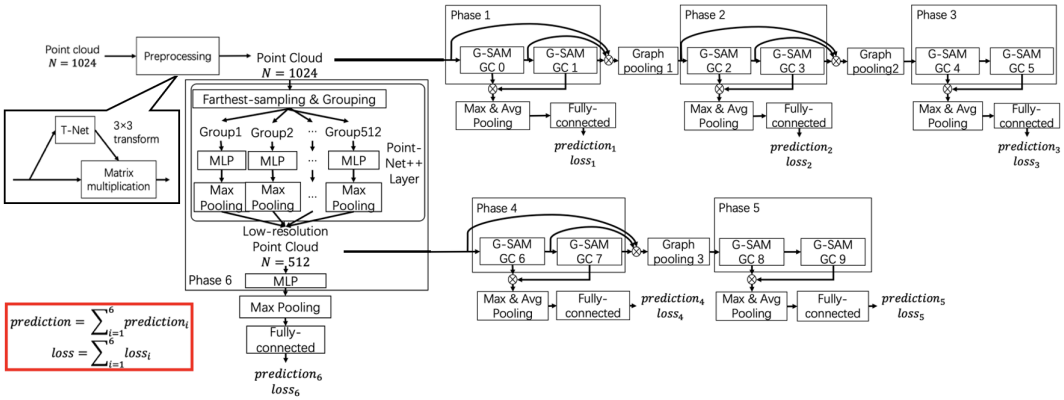

Fig. 4.6. Architecture of the 3D point cloud classification network where $\otimes$ represents the concatenation operation. The model adds up the losses and predictions from the different phases to obtain the overall classification loss and final prediction.

Description

We preprocess the point cloud using the same transformation module (T-Net) as PointNet [68] to lessen rotational effects, which introduced in section 3.4.1. In addition, we build a low-resolution point cloud using a layer of PointNet++ [65], which introduced in section 3.4.1. The multi-resolution structure can easily extract both global and local information from the point cloud.

The specifics of the building blocks created expressly for point cloud classification are described in the sections that follow.

Grouped SAMGC module

Fig. 4.7 depicts the structure of the grouped SAMGC module (G-SAMGC in Fig.4.6). A set of F-dimensional features from the layer before are input of this module.

$\mathcal{H} = \{h_j\}_{j=1}^{M}$ is the input of the module, with M denoting the number of features in the input. For the first module, $\mathcal{H}$ is simply $\mathcal{X}$ and $F = 3$, which represents x, y, and z coordinates of the points.

The k-NN algorithm is used to build the graph for $\mathcal{H}$. We create numerous graphs with various $k = \{11, 41\}$ values to reduce the impact of k in the k-NN. Multiple graph signals are simultaneously analyzed utilizing a collection of ResNeXt [69]-inspired SAMGCs.

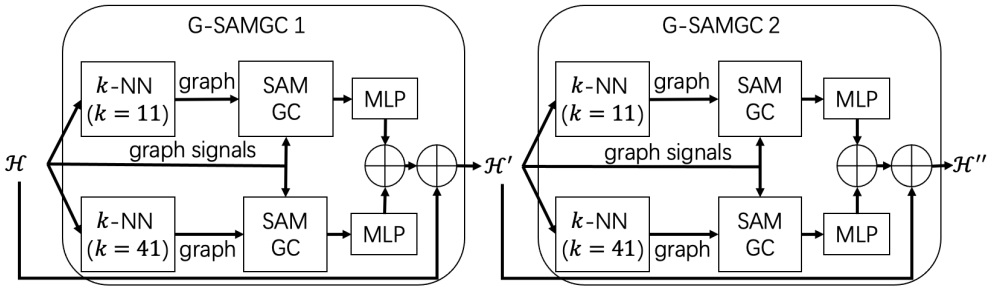

Fig. 4.7. Structure of the grouped SAMGC module. We have two modules (G-SAMGC 1 and G-SAMGC 2), and two k-NNs are applied to the input to create dynamic edges in each module.

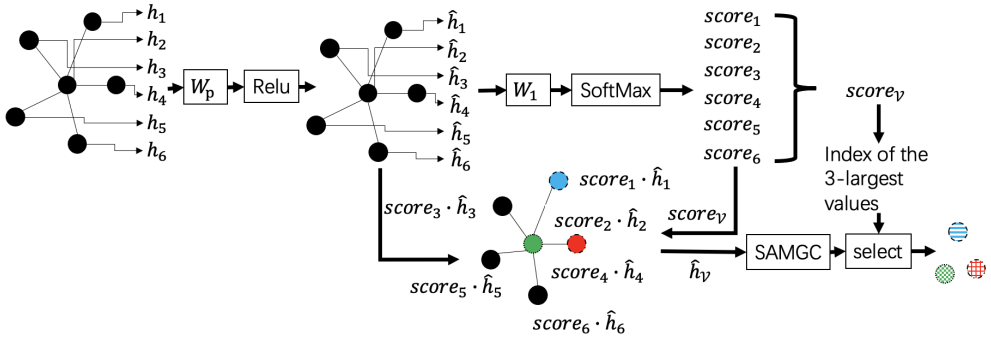

Fig. 4.8. Example of our graph pooling for $w = 3$.

Dynamic graph convolution, which allows the graph topology to change at each layer, outperforms a fixed graph structure, according to [58, 70]. As a result, several graphs are created for various G-SAMGC modules.

Graph Pooling

Effective graph pooling approaches are a popular topic in GNNs and graph signal processing [53, 54]. Early investigations made use of global node representation pooling and graph coarsening methods.

Two trainable graph pooling techniques, DiffPool [59] and GraphU-net [60], were recently introduced. We propose a GraphU-net-inspired score-based graph pooling technique.

Fig. 4.8 depicts our graph pooling. First, we embed node-wise features as follows:

$$\hat{h}_v = \sigma(W_{\mathrm{p}} \cdot h_v), v \in V, \tag{4.14}$$

where W_{p} is the shared learnable weight. We then learn a score on v as follows:

$$\begin{aligned} \mathrm{score}_v &= \mathrm{SoftMax}(W_1 \cdot \hat{h}_v) \\ &:= \frac{\exp\left(W_1 \cdot \hat{h}_v\right)}{\sum_{u \in V} \exp\left(W_1 \cdot \hat{h}_u\right)}, v \in V, \end{aligned} \tag{4.15}$$

where W_1 is the shared learnable weight. Subsequently, features on nodes are updated using the scores, i.e., $\hat{h}_V = \{\mathrm{score}_v \cdot \hat{h}_v\}$.

Subsequently, we update each node-wise features of a node using the SAMGC as follows:

$$\hat{h}'_V = \mathrm{SAMGC}(\hat{h}_V), \tag{4.16}$$

Fig. 4.9. Architecture of the node classification network (i.e., forward propagation), where $\mathcal{G}$ is the graph and $\mathcal{H}$ is the node-wise features.

Finally, we arrange the nodes in descending order according to their scores and select the top w nodes as follows:

$$h_{\text{select}} = \hat{h}'_V[\text{idx}_{\text{select}}] \tag{4.17}$$

where h_{select} is the set of node-wise features in the smaller graph, $\text{idx}_{\text{select}} = \text{rank}\left(\{\text{score}_v\}_V, w\right)$ is the indices of the w nodes selected, in which $\text{rank}\left(\cdot\right)$ is the node ranking operation that returns indices of the w highest scores.

As depicted in Fig. 4.8, we choose w nodes from the learnt scores. We also use the SAMGC to process the features of the chosen nodes, in contrast to GraphU-net, which outputs the selected w nodes directly.

Hierarchical Prediction Architecture

We use the inter-mediate supervision technique [71] and present a hierarchical prediction architecture to properly leverage the hierarchical characteristics (Fig. 4.6). In conclusion, we take into account include the loss at each G-SAMGC module in the total loss function.

There are two G-SAMGC modules in each phase. It was filled using maximum and average pooling layers. These outputs are then concatenated and used as the input for a fully connected layer. We calculate the prediction label and the classification loss for each stage. The losses of several phases are added to the overall classification loss and final prediction labels. This processing can be represented as follows:

$$prediction = \sum_{i=1}^{P} prediction_i, \tag{4.18}$$

$$loss = \sum_{i=1}^{P} loss_i, \tag{4.19}$$

where *prediction* is the final prediction value, $loss_i$ is the cross entropy loss. *loss* is the overall classification loss, and P is the number of phases.

The benefit of this architecture is that by combining the outcomes of the several phases, we can produce predictions that are more dependable and robust.

4.4.2 Node Classification

The overall design of the SAMGC-based node classification network is shown in Fig.4.9. We merely concatenate three SAMGC layers and a fully connected layer to predict the label for each node. The loss function is also the cross entropy loss.

4.5 Experimental Results

4.5.1 Point Cloud Classification

We now provide our experiments for classifying point clouds. We evaluated the performance of proposed GNNs using the ModelNet dataset [66].

Dataset Introduce

A total of 12,308 computer-aided design (CAD) models in 40 categories are available on ModelNet40, including 2,468 for testing and 9,840 for training. 4,899 CAD models total, 3,991 for training and 908 for testing, are included in ModelNet10. We used the same data structure as earlier 3D point cloud categorization techniques: 1,024 points were evenly sampled among the faces of the CAD mesh. Initially, all point clouds were normalized to be in a unit sphere.

Settings and evaluates

The settings of hyper parameters of the point cloud classification network are shown in Table 4.1.

Table 4.1: The hyper parameters of the point cloud classification network. k is the value of k-NN, w is the number of selected nodes. We set t for the multi-hop GC introduced in Section 4.3. For Pointnet++ layer, S is the number of sampled points, r is the radius of each group, D is the number of points of each group.

Learning rate	0.001335			
Drop out	0.3			
t	2			
Graph pooling layername	k	w	[input channels, output channels]	
Graph pooling 0	41	512	[64,64]	
Graph pooling 1	41	128	[128,128]	
Graph pooling 2	41	128	[128,128]	
Graph convolution layername	k	[input channels, output channels]		
G-SAMGC 0	[11,41]	[3,64]		
G-SAMGC 1	[11,41]	[64,64]		
G-SAMGC 2	[11,41]	[64,64]		
G-SAMGC 3	[11,41]	[64,128]		
G-SAMGC 4	[11,41]	[128,256]		
G-SAMGC 5	[11,41]	[256,256]		
G-SAMGC 6	[11,41]	[128,128]		
G-SAMGC 7	[11,41]	[128,128]		
G-SAMGC 8	[11,41]	[128,256]		
G-SAMGC 9	[11,41]	[256,256]		
Pointnet++ layer name	S	r	D	[input channels, output channels]
Pointnet++	512	0.2	32	[3,64]
		0.4	128	[3,64]

We evaluated the accuracy of classification with other networks created specifically for point cloud classification. Owing to the class imbalance in the dataset, we used two performance metrics to assess the results: Average accuracy of all test instances (OA) and average accuracy of all shape classes (mAcc).

Results and explanation

The results are summarized in Table 4.2 for which the scores of the alternative methods were obtained from their corresponding references. We discovered that our network had higher accuracy than the other techniques on both OA and mAcc. We believe that graph-based approaches can provide greater relational information between points when compared to Point-wise MLP methods and Convolution-based methods. In addition, when compared to

Table 4.2: Comparison results of the 3D shape classification on the ModelNet benchmark. OA indicates the average accuracy of all test instances, and mAcc indicates the average accuracy of all shape categories. The symbol "-" indicates that the results are not available from the references.

Type	Method	ModelNet40		ModelNet10	
		OA	mAcc	OA	mAcc
Pointwise MLP Methods	PointNet [68]	89.2%	86.2%	-	-
	PointNet++ [65]	90.7%	-	-	-
	SRN-PointNet++ [82]	91.5%	-	-	-
	PointASNL [83]	93.2%	-	95.9%	-
Convolution-based Methods	PointConv [84]	92.5%	-	-	-
	A-CNN [85]	92.6%	90.3%	95.5%	95.3%
	SFCNN [86]	92.3%	-	-	-
	InterpCNN [87]	93.0%	-	-	-
	ConvPoint [88]	91.8%	88.5%	-	-
Graph-based Methods	ECC [58]	87.4%	83.2%	90.8%	90.0%
	DGCNN [70]	92.2%	90.2%	-	-
	LDGCNN [78]	92.9%	90.3%	-	-
	Hassani et al. [79]	89.1%	-	-	-
	DPAM [72]	91.9%	89.9%	94.6%	94.3%
	KCNet [77]	91.0%	-	94.4%	-
	ClusterNet [80]	87.1%	-	-	-
	RGCNN [73]	90.5%	87.3%	-	-
	LocalSpecGCN [74]	92.1%	-	-	-
	PointGCN [75]	89.5%	86.1%	91.9%	91.6%
	3DTI-Net [76]	91.7%	-	-	-
	Grid-GCN [81]	93.1%	91.3%	97.5%	97.4%
	SAGC	93.5%	91.3%	**98.3%**	97.7%
	SAMGC	**93.6%**	**91.4%**	**98.3%**	**97.8%**

existing Graph-based approaches, our method uses suggested structural features to obtain detailed local structural information in feature space in addition to relational information between points. Our method can therefore have a wider receptive field by leveraging the multi-hop neighborhoods. Finally, by using a multi-resolution point cloud, our method can concurrently use local and global information from the point cloud. As a result, our method can have greater accuracy than previous methods.

4.5.2 Node Classification

We also ran experiments on a node classification task on Cora [67] and Pubmed [89], to further validate the performance of SAMGC.

Dataset Introduce

Standard citation network datasets include Cora [67] and Pubmed [89]. The nodes of datasets stand in for papers, while their edges stand in for citations. Table 4.3 displays the details of Cora [67] and Pubmed [89].

Table 4.3: The details of the Cora and Pubmed benchmark.

Items	Cora	Pubmed
The number of nodes	2,708	19,717
The number of edges	5,429	44,338
The dimension of features	1,433	500
The number of classes	7	3

Table 4.4: The hyper parameters of node classification network. We set t for the multi-hop GC introduced in Section 4.3.

Learning rate	0.1	
Drop out	0.3	
t	2	
layername	[input channels, output channels]	
	Cora	Pubmed
SAMGC 0	[1433,128]	[500,128]
SAMGC 1	[128,256]	[128,256]
SAMGC 2	[256,1024]	[256,1024]

Settings and evaluates

The hyper parameters of the node classification network are shown in Table 4.4. We use the classification accuracy to validate the performance.

Results and explanation

The classification outcomes utilizing representative methods are displayed in Table 4.5. Despite the simplicity of the structure of our network, we found that it was extremely accurate at classifying citation networks. In contrast to earlier graph neural networks, our method may first collect thorough local structural data in the feature space. Then, by using multi-hop convolution, our method may be able to collect crucial information in indirect neighborhoods as well. The proposed one-hop Neighbor-wise learnable average aggregation can finally fully exploit the data from the one-hop neighborhood. Our method has a greater classification accuracy since the processed nodes store richer information compared to other approaches.

In the following section, the effectiveness of several SAMGC components is demonstrated.

4.5.3 Effect of SAMGC

In this section, we conduct experiments to confirm the effect of SAMGC.

Point cloud classification

We conducted experiments on the point classification task for ModelNet [66] utilizing several spatial GCs in order to verify the efficacy of various SAMGC components as follows:

1. **SAGConv**. We picked the SAGConv that described in chapter 3 as the baseline.
2. **Neighbor-wise Learnable Average Aggregation SAGConv**. We replaced the average aggregation procedure of SAGConv with the neighbor-wise learnable average aggregation method proposed in section 4.3.2.

Table 4.5: Node classification results for the Cora and Pubmed benchmark. The symbol "-" indicates that the results are not available from the references.

Method	Accuracy	
	Cora	Pubmed
CayleyNet [33]	81.2%	-
GCN [19]	81.4%	79.0%
GraphSAGE [29]	82.1%	76.2%
GAT [30]	83.0%	79.3%
SAGConv	83.0%	79.4%
Ours	**86.4%**	**80.2%**

3. **Neighbor-wise Learnable Average Aggregation SAGConv**. In addition, we replaced the average aggregation procedure of GraphSAGE [29] with the neighbor-wise learnable average aggregation method proposed in section 4.3.2.

4. **SAGConv-2hop**. In this GC, we utilized 2-hop neighboring nodes of a target node.

5. **SAMGC**. The complete SAMGC as mentioned in section 4.3.

Table 4.6 shows their details. The architecture of network and the settings are the same as

Table 4.6: Different methods of spatial GCs. "sf" represents structural features which are described in section 4.3.1. "nwa($\cdot$)" represents neighbor-wise learnable average aggregation operation, which is described in Section 4.3.2.

Method	Aggregation operation	Integration operation
SAGConv	$h_{N_1(v)} = \frac{1}{\lvert N_1(v)\rvert}\sum_{u\in N_1(v)} \mathrm{cat}(h_u, sf)$	$h'_v = \sigma(W \cdot \mathrm{cat}(h_v, h_{N_1(v)}))$
Neighbor-wise Learnable Average Aggregation SAGConv	$h_{N_1(v)} = \mathrm{nwa}(h_u, sf)$	$h'_v = \sigma(W \cdot \mathrm{cat}(h_v, h_{N_1(v)}))$
SAGConv-2hop	$h_{N_1(v)} = \frac{1}{\lvert N_1(v)\rvert}\sum_{u\in N_1(v)} \mathrm{cat}(h_u, sf)$ $\mathrm{nh}_{N_2(v)} = \frac{1}{\lvert N_2(v)\rvert}\sum_{u\in N_2(v)} h_u$	$h'_v = \sigma(W \cdot \mathrm{cat}(h_v, h_{N_1(v)}, \mathrm{nh}_{N_2(v)}))$
SAMGC	$h_{N_1(v)} = \mathrm{nwa}(h_u, sf)$ $\mathrm{nh}_{N_2(v)} = \frac{1}{\lvert N_2(v)\rvert}\sum_{u\in N_2(v)} h_u$	$h'_v = \sigma(W \cdot \mathrm{cat}(h_v, h_{N_1(v)}, \mathrm{nh}_{N_2(v)}))$

we used in section 3.5.2.

The results are summarized in Fig. 4.10. We saw that the SAGConv result was 92.1% . The accuracy of neighbor-wise learnable average aggregation SAGConv was raised by 0.1% . The accuracy of SAGConv-2hop was raised by 0.4% . The accuracy of the SAMGC was raised by 0.5% . This study provided evidence for the usefulness of SAMGC modules.

Node classification

We conducted experiments on a node classification task for Cora [67] utilizing several spatial GCs in order to verify the efficacy of various SAMGC components as follows:

1. **GraphSAGE [29]**. GraphSAGE [29], as we described in section 4.2.2, is a typical approach for spatial GC. We picked this as the baseline because it just makes use of the node-wise characteristics of the one-hop neighborhood.

2. **Structure-aware GC (SAGConv)**. To validate the effectiveness of structural features, we utilize the spatial GC which proposed in chapter 3.

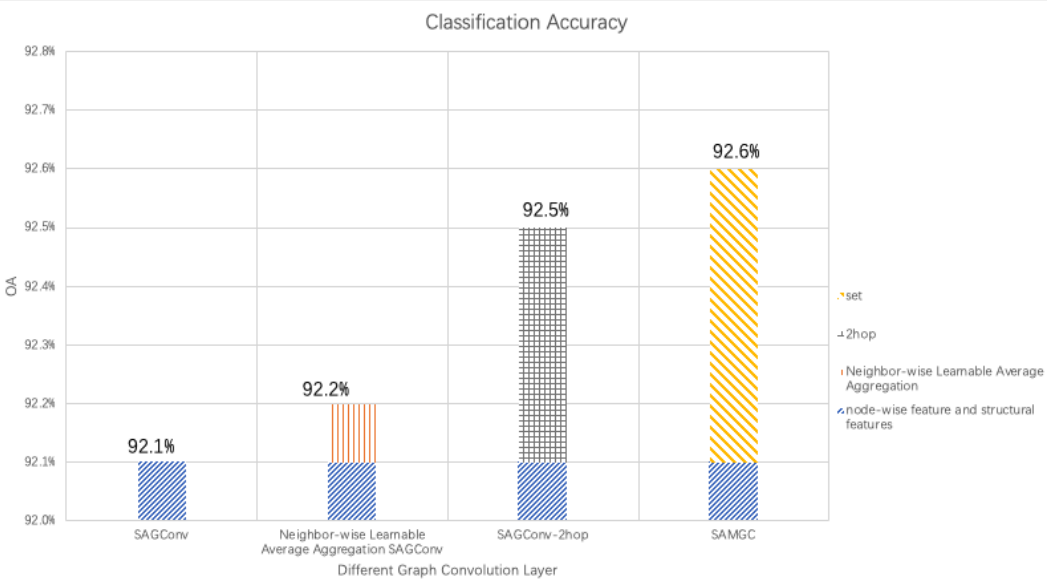

Fig. 4.10. Point cloud classification results for the ModelNet benchmark using different spatial GCs.

3. **Neighbor-wise Learnable Average Aggregation SAGConv**. In addition, we replaced the average aggregation procedure of GraphSAGE [29] with the neighbor-wise learnable average aggregation method proposed in section 4.3.2.

4. **SAMGC**. The complete SAMGC as mentioned in section 4.3.

Table 4.7 shows their details.

Table 4.7: Different methods of spatial GCs. "sf" represents structural features which are described in section 4.3.1. "nwa(·)" represents neighbor-wise learnable average aggregation operation, which is described in Section 4.3.2.

Method	Aggregation operation	Integration operation
GraphSAGE [29]	$h_{N_1(v)} = \frac{1}{\lvert N_1(v)\rvert}\sum_{u\in N_1(v)} h_u$	$h'_v = \sigma(W \cdot \mathrm{cat}(h_v, h_{N_1(v)}))$
SAGConv	$h_{N_1(v)} = \frac{1}{\lvert N_1(v)\rvert}\sum_{u\in N_1(v)} \mathrm{cat}(h_u, sf)$	$h'_v = \sigma(W \cdot \mathrm{cat}(h_v, h_{N_1(v)}))$
Neighbor-wise Learnable Average Aggregation SAGConv	$h_{N_1(v)} = \mathrm{nwa}(h_u, sf)$	$h'_v = \sigma(W \cdot \mathrm{cat}(h_v, h_{N_1(v)}))$
SAMGC	$h_{N_1(v)} = \mathrm{nwa}(h_u, sf)$ $\mathrm{nh}_{N_2(v)} = \frac{1}{\lvert N_2(v)\rvert}\sum_{u\in N_2(v)} h_u$	$h'_v = \sigma(W \cdot \mathrm{cat}(h_v, h_{N_1(v)}, \mathrm{nh}_{N_2(v)}))$

The results are summarized in Fig. 4.11. We saw that the GrphSAGE [29] result was 82.0% . The accuracy of the SAMGC was raised by 2.3% , the neighbor-wise learnable average aggregation by 1.1% , and the result by 1.0% thanks to the structural features. This study also provided evidence for the usefulness of SAMGC modules.

4.6 Conclusion

In this study, we introduce the SAMGC, a novel spatial graph convolution method. It determines relational embedding, feature angles, and distances in the feature space. Additionally, it gathers data from multi-hop neighbors nodes within one step of the GC. We offer GNNs

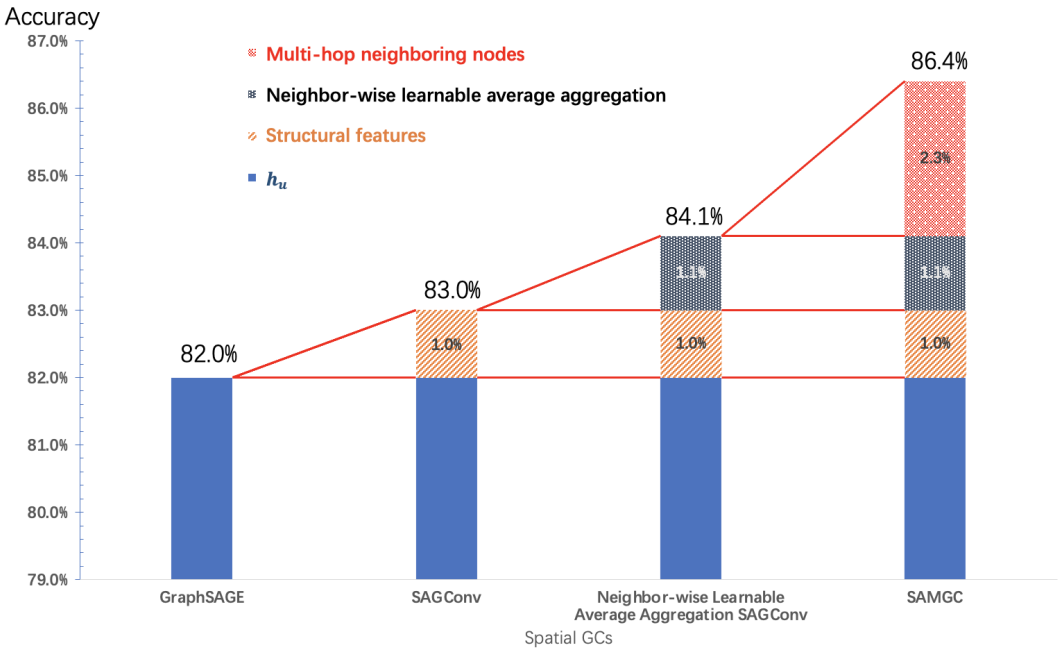

Fig. 4.11. Node classification results for the Cora benchmark using different spatial GCs.

based on the SAMGC for graph and node classification. Our method performs better than other ways, according to experiments. As a result, we showed how GC performance may be enhanced by local structural information in feature space as well as relevant information from indirect neighborhoods. How to better utilize the local structure information in the feature space and the relevant information in the indirect neighbors during the graph convolution process are the two main areas of interest in our subsequent research.

Chapter 5

MTSPAGC: Multi-type Structure and position-aware attention-based Graph Convolution

5.1 Introduction

Due to vast datasets and improvements in computer power, deep neural networks have achieved exceptional success in identifying, segmenting, and recognizing regularly structured data such as photos and videos [15–17]. It is difficult to process many irregular data sets without a fixed sample order in the real world for traditional deep learning techniques. Such examples include attributes on 3D point clouds, opinions on social networks, and the number of passengers in traffic networks. Transforming irregularly structured data into graph-structured data could improve the processing of such data. Graph neural networks (GNNs) have gotten a lot of interest in this context [25, 44, 45].

In existing GNNs, graph convolutions (GCs), often referred to as graph filters, are usually used as a time and spatial domain counterpart of convolution in existing GNNs. The main idea of GC algorithms is to iteratively aggregate features from neighboring nodes before integrating the aggregated information with that of the target node [25, 26, 63]. Existing GCs occur over-smoothing. Over-smoothing makes node-wise features of nodes where the edge exists be same [25, 28]. Over-smoothing may influence the performance of GNNs. The frequency of over-smoothing happenings can be reduced by improving the expressive of GCs [64].

On the other hand, the attention mechanism has been widely utilized in many tasks, such as machine translation [49–51], machine reading [52], and so on. Therefore, introducing the attention mechanism to graph convolution is also be considered. These methods are called attention-based graph convolutions (AGCs). By treating neighboring nodes differently according to attention weights, AGCs can significantly improve the expressive of GCs. However, we still think two limitations of expressive exist in AGCs:

1. Existing methods mostly use node-wise features of the one-hop neighborhood in a single step of GCs. Therefore, existing methods might lose the *structural information* of the surrounding neighboring nodes, particularly in the feature space.

2. When calculating the attention weight for each neighbor node, one type of attention function is utilized, i.e., concatenation, subtraction, or multiplication. However, differ-

ent types of attention functions leads to different attention weights which may affect the expressive of methods. Existing methods do not consider the influence of the type of attention function on the attention weights when only using one type of attention function.

In other words, if we can effectively 1) utilize the intricate structural details of the surrounding neighboring nodes in the feature space and 2) eliminate the influence of the type of attention function on the attention weights, the expressive of AGCs may be further enhanced.

In this chapter, we propose a new AGC addressing the above two challenges. Our proposed AGC has two methods.

First, we utilize feature angle, feature distance, and relational embedding, which proposed in chapter 3, as three structural characteristics to describe the surrounding structure in the feature space. They are combined with node-wise features.

Second, we propose a multi-type attention function. Different from the existing attention-based GNNs, in our method, two types of attention functions are utilized simultaneously to calculate the attention weights.

The two methods can be used simultaneously to get our multi-type structure and position-aware attention-based Graph Convolution (MTSPAGC). The following is a summary of our contributions:

1. We can extract detailed structural information on surrounding neighboring nodes in the feature space. It allows for that the convolved nodes contain richer information than existing techniques.

2. By using multi-type attention functions simultaneously, we can obtain better attention weights than only use one type of attention function.

3. We build GNNs for graph classification tasks and node classification tasks based on the MTSPAGC. For graph classification, 3D point clouds are used as a representative example. MTSPAGC layers are combined with a PointNet++ [65] layer and graph pooling layers. Experiments on the ModelNet [66] dataset showed that our method has higher classification accuracies than previous methods. For node classification, part segmentations of 3D point clouds are used as a representative example. MTSPAGC layers are combined with a PointNet++ [65] layer, graph pooling layers and feature propagation layers which are used in PointNet++ [65]. Experiments on the ShapeNet [90] dataset showed that our method also has a good performance.

Notation: An undirected graph is defined as $G = (V, E)$ where V is a set of nodes, and E is a set of edges. The adjacency matrix of G is denoted as A. $\widetilde{D}$ is the diagonal degree matrix.

Here, $h_v := [\mathrm{h}_{v1}, \ldots, \mathrm{h}_{vj}, \ldots, \mathrm{h}_{vC}]^\top \in \mathbb{R}^C$ represents a feature vector on the node $v \in V$, and C is the number of features in h_v.

The non-linearity function is denoted as $\sigma(\cdot)$. The number of neighborhood hops is t. The set of one-hop neighboring nodes is $N(\cdot)$ in which its cardinality is denoted as $|N(\cdot)|$. A multilayer perceptron (MLP) layer is represented as $\mathrm{MLP}(\cdot)$. A channel-wise max-pooling is denoted as $\mathrm{MaxPool}(\cdot)$. The vector concatenation operation is denoted as $\mathrm{cat}(\cdot)$.

5.2 Preliminaries

The fundamental idea behind GCs is to incrementally aggregate the data of neighbor nodes and then integrate the aggregated data with that of the target node [25]. The two types of GC methods now in use are spectral and spatial methods. Furthermore, the spatial methods can be divided into simple spatial methods and attention-based spatial methods.

5.2.1 Spectral methods

For spectral approaches, a GC operation is carried out in the graph Fourier domain. The graph Fourier basis is the eigenvector matrix of the graph Laplacian [15,31]. Polynomial approximations are typically employed in place of eigendecomposition since the computational complexity of it is high when processing large scale graphs [18,32,33].

In order to further simplify the algorithm, a linear graph filter is introduced in [19]. It is

$$H_{GCN} = \sigma(\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}HW), \tag{5.1}$$

where $H := \{h_v\}_{v\in\mathcal{V}}$ is the set of node-wise features, $\widetilde{A} = A + I_n$ is the adjacency matrix with self-loops, and W is the trainable weight matrices. This is based on the presumption that the spectral filter is a linear function of the graph frequency, i.e. eigenvalue, is true. GCN is the most widely utilized spectral method.

5.2.2 Simple spatial methods

In the target node, simple spatial GC algorithms directly aggregate the node-wise features of one-hop neighboring nodes. A simple spatial methods include aggregation and integration operations. It can be represented as follows:

$$h'_v = \text{integration}(h_{N(v)}, h_v), v \in \mathcal{V} \tag{5.2}$$

where

$$h_{N(v)} = \text{aggregation}(h_u), u \in N(v) \tag{5.3}$$

in which h'_v is the updated node-wise feature of the target node v. Furthermore, $h_{N(v)}$ is the aggregated node-wise feature at $N(v)$. The aggregation operation gathers and aggregates the features of neighboring nodes. Aggregation by average or sum is frequently employed. Finally, during the integration operation, features are marged into the target node.

GraphSAGE [29] is a typical general framework of normal spatial methods, each neighboring node is handled identically, it employs an Average aggregator, LSTM aggregator or Pooling aggregator.

5.2.3 Attention-based spatial methods

As we mentioned before, to efficiently utilize the information of each neighboring node, the self-attention or others have been introduced into spatial GCs. Instead of treating each neighboring nodes equally, attention-based spatial methods calculate an attention weight for each neighboring node. Then, attention-based methods use the weighted sum to aggregate features of neighboring nodes.

Graph attention network [30] is the first introduce attention mechanism to graph convolution operation. The hidden embedding of node v can be obtained as follow:

$$\begin{aligned} \mathbf{h}_v &= \sigma(\sum_{u\in N(v)} a_{vu}\mathbf{W}\mathbf{h}_u), \\ a_{vu} &= \frac{exp(LeakyReLU(\mathbf{a}^{\mathrm{T}}[\mathbf{W}\mathbf{h}_v||\mathbf{W}\mathbf{h}_u]))}{\sum_{i\in N(v)} exp(LeakyReLU(\mathbf{a}^{\mathrm{T}}[\mathbf{W}\mathbf{h}_v||\mathbf{W}\mathbf{h}_i]))}. \end{aligned} \tag{5.4}$$

where $\mathbf{a}$ is the weight vector of a single-layer MLP and $\mathbf{W}$ is the weight matrix associated with the linear transformation that is applied to each node. $||$ is the concatenation operation. We can see that GAT use concatenation operation in the attention function rather than scaled dot-product operation.

GAT also utilizes multi-head attention. It uses K independent attention head to compute the hidden states and concatenates them or compute the average of them. Output representations are shown as follows:

$$\begin{aligned} \mathbf{h}_v &= ||_{k=1}^{K} \sigma(\sum_{u \in N(v)} a_{vu}^k \mathbf{W}\mathbf{h}_u), \\ \mathbf{h}_v &= \sigma(\frac{1}{K}\sum_{k=1}^{K}\sum_{u \in N(v)} a_{vu}^k \mathbf{W}\mathbf{h}_u). \end{aligned} \tag{5.5}$$

Here the result of the kth attention head a_{vu}^k is the normalized attention coefficient.

Based on GAT, to improve the performance for noisy graph, SuperGAT [36] was proposed.

Recently, Transformer [51] was proposed for NLP tasks. The Transformer [51] first constructs a complete graph for the input. Each node weighted adds and integrates features of other nodes. It utilizes the self-attention mechanism to calculate attention weights. According to the concept of spatial graph convolution, we think Transformer [51] is also an attention-based spatial method. In the following, we introduce the original Transformer [51].

In 2017, Transformer [51] was proposed, and it has gotten a remarkable performance. Recently, it has been utilized not only in NLP but also in computer vision [91]. The Transformer [51] has two parts, the self-attention part, and the feed-forward network part.

The self-attention part can be represented as follows:

$$\begin{aligned} Z_{att} &= f_{att}(Q, K)V, \\ Q &= W_q \cdot X, \\ K &= W_k \cdot X, \\ V &= W_v \cdot X, \end{aligned} \tag{5.6}$$

where, X is the set of input, W_q, W_k, W_v are learnable weights, $f_{att}(\cdot)$ is the attention function which calculate attention weights, Z_{att} is the output of self-attention part. In the Transformer [51], the scaled dot-product is utilized as attention function as follows:

$$f_{att} = Softmax(\frac{QK^T}{\sqrt{d_k}}), \tag{5.7}$$

where d_k is the number of dimensions of K.

The feed-forward network part can be represented as follows:

$$Z = W_2(\sigma(W_1 Z_{att} + b_1)) + b_2, \tag{5.8}$$

where W_1 and W_2 are learnable weights, b_1 and b_2 are biases, Z is the output.

To better represent the relationship, multi-head attention aims to attend to information from different sub-spaces. Multi-head attention has h scaled dot-product attention modules and get h different outputs:

$$Z_{atti} = f_{att}(W_{qi} \cdot X, W_{ki} \cdot X)(W_{vi} \cdot X), i = 1, 2, ...h, \tag{5.9}$$

where W_{qi}, W_{ki}, W_{vi} are learnable weights for ith head, Z_{atti} is the output of ith head. Finally, the h outputs are concatenated and inputed into a linear part to get the final output. In the Transformer, self-attention part is the most important part.

In the GraphFormers [37], GNNs and Transformer are connected. To make Transformer more fit to graph data, Graphormer [38] introduces graph structure embedding into Transformer.

In contrast to previous methods, we first install structure features that represent the structure of surrounding neighboring nodes in the feature space into an attention-based

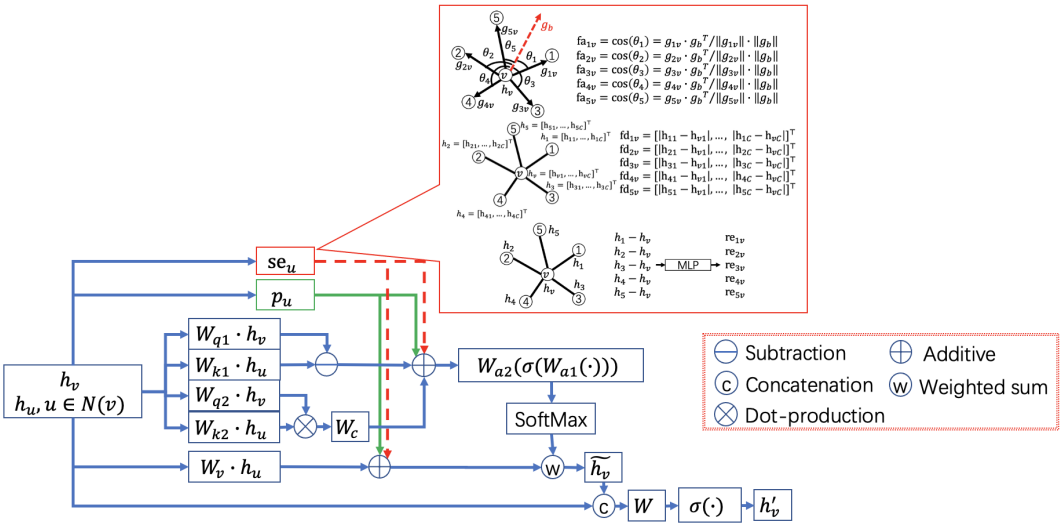

Fig. 5.1. The over-all structure of MTSPAGC.

spatial method. Second, we simultaneously use different types of attention functions to calculate attention weights. We simultaneously employ both of these two methods for our new attention-based spatial method, MTSPAGC.

5.3 MTSPAGC

In this section, we introduce our MTSPAGC. It is shown in Fig.5.1. As mentioned before, we have two goals:

1) We utilize the structural information of surrounding neighboring nodes in the feature space during a single step GC;

2) We eliminate the influence of the type of attention function on the attention weights using multi-type attention functions simultaneously.

To achieve these, the MTSPAGC has five parts:

1. Structural features;
2. Structure embedding;
3. Position embedding;
4. Weighted sum aggregation;
5. Integration.

Algorithm 3 shows its details, and we sequentially introduce the five parts.

5.3.1 Structural features

We use three structural features in the feature space, which proposed in chapter 3 to obtain precise structural information of the surrounding neighboring nodes:

- Feature angle;
- Feature distance;
- Relational embedding.

Below, we simply review these three features.

Algorithm 3 MTSPAGC spatial graph convolution (i.e., forward propagation) algorithm

Input: graph $\mathcal{G} = (V, E)$; input features $\{h_v, v \in V\}$; weight matrices W_{nw}, W_{SE}, W_{PE1}, W_{PE2}, W_{q1}, W_{k1}, W_{q2}, W_{k2}, W_{att1}, W_{att2}, W_v and W_l; non-linearity function $\sigma(\cdot)$; neighborhood function $N(\cdot)$; MLP layer MLP$(\cdot)$; channel-wise max-pooling MaxPool$(\cdot)$; concatenation operation cat$(\cdot)$; transposition $\cdot^{\mathrm{T}}$; SoftMax operation SoftMax$(\cdot)$

Output: Convolved features z_v for all $v \in V$

for $v \in \mathcal{V}$ **do**

$\mathcal{F}_{N(v)} = \{g_{uv} := h_u - h_v\}$

$g_b = \mathrm{MaxPool}(\{\sigma(\mathrm{MLP}(g_{uv}))\})$

$\mathrm{fa}_{uv} = \cos(\theta_u) = \dfrac{g_{uv} \cdot g_b^{\mathrm{T}}}{\|g_{uv}\| \cdot \|g_b\|}, g_{uv} \in \mathcal{F}_{N(v)}$

$\mathrm{fd}_{uv} = [|\mathrm{h}_{u1} - \mathrm{h}_{v1}|, ..., |\mathrm{h}_{uC} - \mathrm{h}_{vC}|]^{\mathrm{T}}$

$\mathrm{re}_{uv} = \sigma(\mathrm{MLP}(h_u - h_v)), u \in N(v)$

$\mathrm{nw}_u = \sigma(W_{nw} \cdot \mathrm{cat}(h_u, \mathrm{fa}_{uv}, \mathrm{fd}_{uv}, \mathrm{re}_{uv})), u \in N(v)$

$\mathrm{se}_u = W_{SE} \cdot \mathrm{cat}(h_u, \mathrm{fa}_{uv}, \mathrm{fd}_{uv}, \mathrm{re}_{uv}, \mathrm{nw}_u), u \in N(v)$

$p_u := h_u - h_v$

$p_u = \sigma(W_{PE1} \cdot (p_u)), u \in N(v)$

$p_u = W_{PE2} \cdot (p_u), u \in N(v)$

$q_{1v} = W_{q1} \cdot h_v$

$k_{1u} = W_{k1} \cdot h_u, u \in N(v)$

$q_{2v} = W_{q2} \cdot h_v$

$k_{2u} = W_{k2} \cdot h_u, u \in N(v)$

$att_{vu} = q_{1v} - k_{1u} + W_c \cdot (q_{2v} \cdot k_{2u}^{\mathrm{T}}) + \mathrm{se}_u + p_u$

$att_{vu} = \mathrm{SoftMax}(W_{att2}(\sigma(W_{att1} \cdot att_{vu})))$

$v_u = W_v \cdot h_u, u \in N(v)$

$z_v = \sum_{u \in N(v)} att_{vu} \cdot (v_u + \mathrm{se}_u + p_u)$

$h'_v = \sigma\left(W_l \cdot \mathrm{cat}\left(h_v, z_v\right)\right)$

end for

$z_v = h'_v, \forall v \in V$

return z_v

Feature angle

In order to explain the structure of the neighboring neighborhoods, we first propose feature angles. We create a set of vectors pointing from v to the nearby neighboring nodes in the feature space: $\mathcal{F}_{N(v)} = \{g_{uv} := h_u - h_v\}_{u \in N(v)}$. Then, a basis vector g_b is learned from $\mathcal{F}_{N(v)}$ as follows:

$$g_b = \mathrm{MaxPool}(\{\sigma(\mathrm{MLP}(g_{uv}))\}_{g_{uv} \in \mathcal{F}_{N(v)}}) \tag{5.10}$$

It is shown in Fig.5.2

Finally, the cosine of the angle between g_{uv} and g_b is computed as follows:

$$\mathrm{fa}_{uv} = \cos(\theta_u) = \frac{g_{uv} \cdot g_b^{\mathrm{T}}}{\|g_{uv}\| \cdot \|g_b\|}, \ g_{uv} \in \mathcal{F}_{N(v)} \tag{5.11}$$

The example is depicted in Fig. 5.3 (a).

Feature distance

The absolute difference between the elements of h_u and h_v is the feature distance between neighboring nodes u and target node v in the feature space. It has the following representation:

$$\mathrm{fd}_{uv} = [|\mathrm{h}_{u1} - \mathrm{h}_{v1}|, ..., |\mathrm{h}_{uC} - \mathrm{h}_{vC}|]^{\mathrm{T}}. \tag{5.12}$$

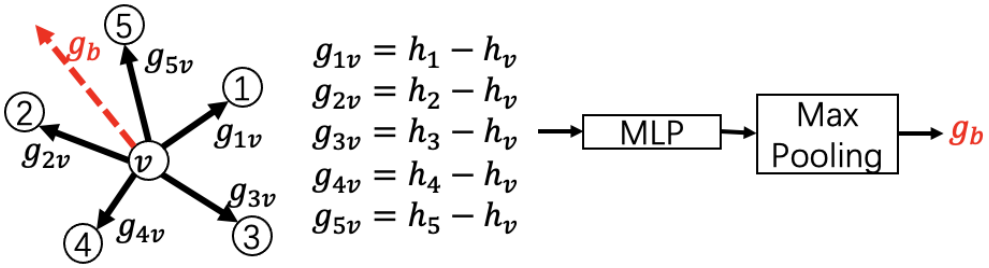

Fig. 5.2. Learning a base vector g_b. The numbers in the black circle are the node indices.

The example is shown in Fig. 5.3 (b).

Relational embedding

Relational embedding illustrates the strength of the influence of one-hop nearby nodes u on the target node v. We can infer it from the following from the distinction between h_v and h_u:

$$\mathrm{re}_{uv} = \sigma(\mathrm{MLP}\,(h_u - h_v)),\ u \in N(v). \tag{5.13}$$

re_{uv} is depicted in Fig. 5.3 (c).

5.3.2 Structure embedding

To fully utilize neighborhood information, we embedding the structure of each neighboring node u in the feature space. Inspired by DensNet [92], the structure embedding has two phases.

In the first phase, we integrate various structural features from section 5.3.1 and node-wise features using learnable weights W_{nw}. This process is described below:

$$\mathrm{nw}_u = \sigma(W_{nw} \cdot \mathrm{cat}(h_u, \mathrm{fa}_{uv}, \mathrm{fd}_{uv}, \mathrm{re}_{uv})), u \in N(v). \tag{5.14}$$

This process is also shown in Fig.5.4

In the second phase, we first concatenate the node-wise features of node u, structural features of node u, and the outputs of the first phase. Then, we use a linear transformation to get the structure embedding of neighboring node u. This process is summarized as follows:

$$\mathrm{se}_u = W_{SE} \cdot \mathrm{cat}(h_u, \mathrm{fa}_{uv}, \mathrm{fd}_{uv}, \mathrm{re}_{uv}, \mathrm{nw}_u), u \in N(v). \tag{5.15}$$

where se_u is the structure embedding of neighboring node u, W_{SE} are learnable weights.

5.3.3 Position embedding

Position encoding is crucial for self-attention machine because it enables the Transformer operator to adapt to local data structure [51]. In existing methods [51,93], position encoding is manual works, and it is always based on sine and cosine functions. Here, we learn a position embedding to make our MTSPAGC better adapts to the local data structure in the feature space, rather than using sine and cosine functions-based position encoding as existing methods.

In the high-dimensional feature space, the node-wise features of a node are also coordinates of a node. We embed the difference of the coordinates between the neighboring node u and target node v in the high-dimensional feature space. Our position embedding p_u for node u is defined as follows:

$$p_u = W_{PE2} \cdot (\sigma(W_{PE1} \cdot (h_u - h_v))), u \in N(v) \tag{5.16}$$

where W_{PE1} and W_{PE2} are learnable weights.

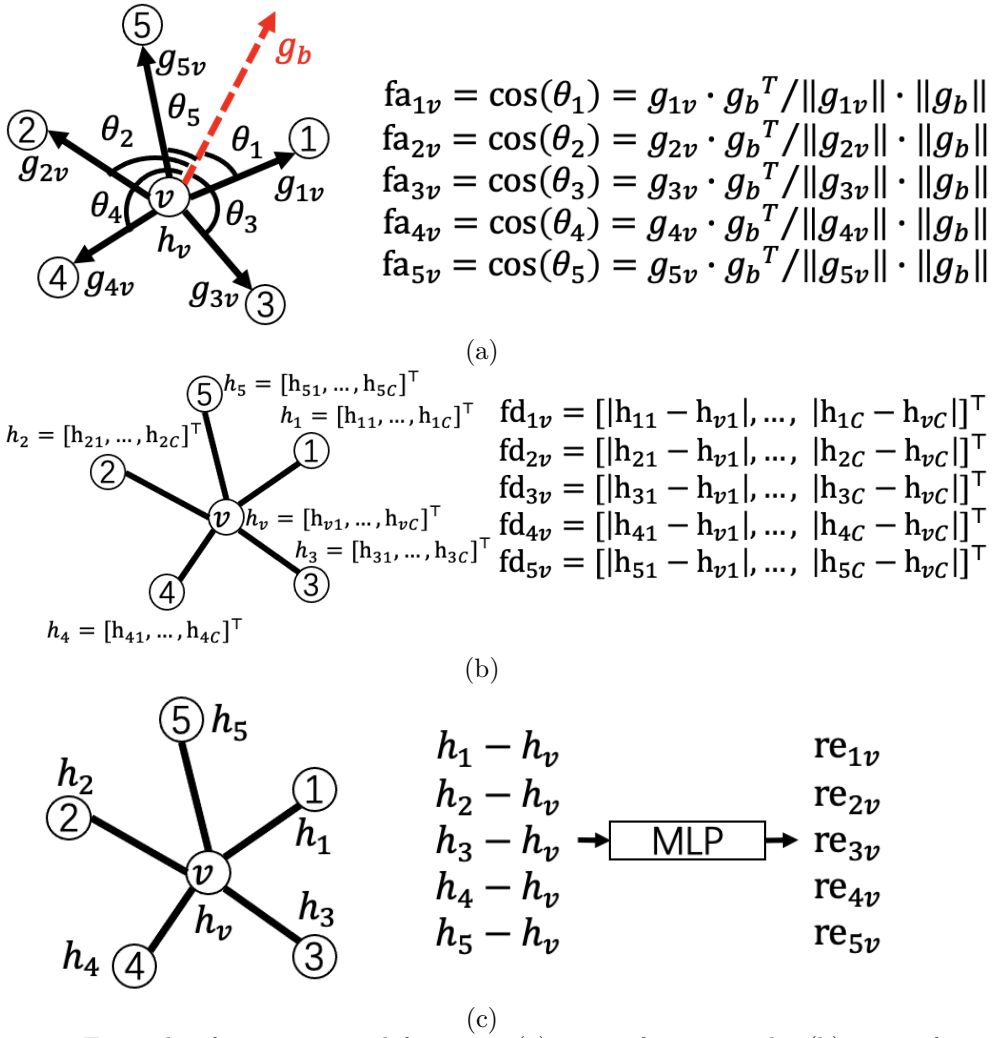

Fig. 5.3. Example of our structural features. (a) is our feature angle; (b) is our feature distance, h_{vj} is the element of h_v, C is the number of elements; (c) is our relational embedding.

5.3.4 Weighted sum aggregation

In this section, we introduce our weighted sum aggregation. We first introduce calculation steps of attention weights. Then, we introduce the weighted sum aggregation.

Attention weights

As we mentioned before, the type of attention function may influence the final result. To further explain the problem, we use the example which is shown in Fig.5.5. In this example, we have a 3D point cloud, i.e., $X = \{x_1(5,0,0), x_2(1,0,0), x_3(1,1,0)\}$. When we use dot-product attention function to calculate the attention weights, we can see that the attention weight (a_{21}) between $x_2(1,0,0)$ and $x_1(5,0,0)$ is 0.964, but the attention weight (a_{23}) between $x_2(1,0,0)$ and $x_3(1,1,0)$ is 0.018. However, in the 3D point cloud, $x_3(1,1,0)$ is more important than $x_1(5,0,0)$ for $x_2(1,0,0)$. Therefore, the dot-product attention function may not suit for processing 3D point clouds.

To deal with this problem, we first propose a multi-type attention function mechanism.

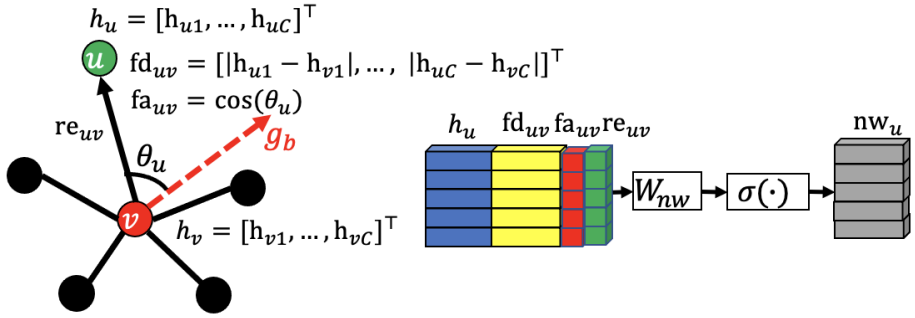

Fig. 5.4. The first phase of structure embedding. The node-wise feature of the 1st-hop neighboring node u is h_u and its corresponding structural features, fa_{uv}, fd_{uv}, and re_{uv}.

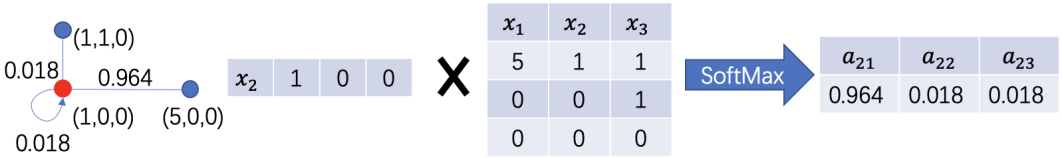

Fig. 5.5. The example of the influence of the type of attention function on attention weights. Here, we have a point cloud consisting of three points in 3D space. When we use dot-production to calculate the attention weights, there is a large attention weight between the point $x_2(1,0,0)$ and the point $x_1(5,0,0)$. However, in the 3D space, the point $x_3(1,1,0)$ is more similar to the point $x_2(1,0,0)$.

In this mechanism, multi-type attention functions are utilized simultaneously to calculate better attention weights than only using one type of attention function.

Subtraction attention function and dot-production attention function are usually utilized in existing methods [51, 93, 94]. We also utilize these two types of attention functions.

Subtraction attention function The first type of attention function is the subtraction attention function, it is defined as follows:

$$\begin{aligned} a_{1vu} = f_1(\mathrm{q}_{1v}, \mathrm{k}_{1u}) &:= \{\mathrm{q}_{1v} - \mathrm{k}_{1u}\}_{u \in N(v)}, \\ \mathrm{q}_{1v} &= W_{q1} \cdot h_v, \\ \mathrm{k}_{1u} &= W_{k1} \cdot h_u. \end{aligned} \tag{5.17}$$

where, W_{q1} and W_{k1} are learnable weights. The example is shown in Fig.5.6.

Dot-production attention function The second type of attention function is the dot-production attention function, it is defined as follows:

$$\begin{aligned} a_{2vu} = f_2(\mathrm{q}_{2v}, \mathrm{k}_{2u}) &:= \{\mathrm{q}_{2v} \cdot \mathrm{k}_{2u}{}^{\mathrm{T}}\}_{u \in N(v)}, \\ \mathrm{q}_{2v} &= W_{q2} \cdot h_v, \\ \mathrm{k}_{2u} &= W_{k2} \cdot h_u. \end{aligned} \tag{5.18}$$

where, W_{q2} and W_{k2} are learnable weights. The example is shown in Fig.5.7.

Finally, we add up outputs of these two types of attention functions as follows:

$$\mathrm{qk}_{vu} = a_{1vu} + W_c \cdot a_{2vu}, u \in N(v) \tag{5.19}$$

where W_c are learnable weights that also converts a_{2vu} into the same dimensions as a_{1vu}.

Then we introduce the structure embedding and the position embedding into the multi-type attention function mechanism. Here, we add up the qk_{vu}, the structure embedding se_u, and the position embedding p_u, then input it into a feed-forward network as follows:

$$\begin{aligned} \mathrm{qk}_{vu}{}' &= \mathrm{qk}_{vu} + \mathrm{se}_u + p_u, u \in N(v), \\ \mathrm{qk}_{vu}{}'' &= W_{att2} \cdot (\sigma(W_{att1} \cdot (\mathrm{qk}_{vu}{}'))), u \in N(v), \end{aligned} \tag{5.20}$$

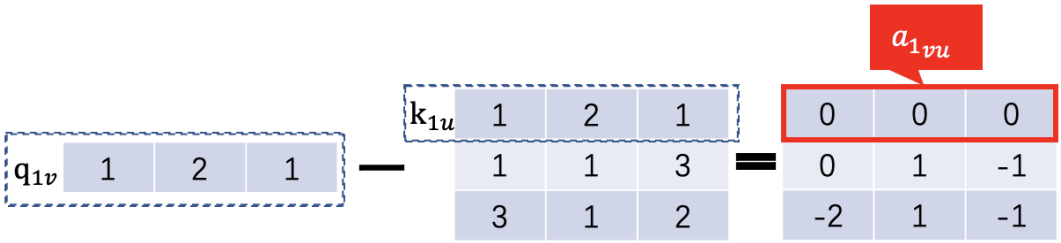

Fig. 5.6. The example of the subtraction attention function.

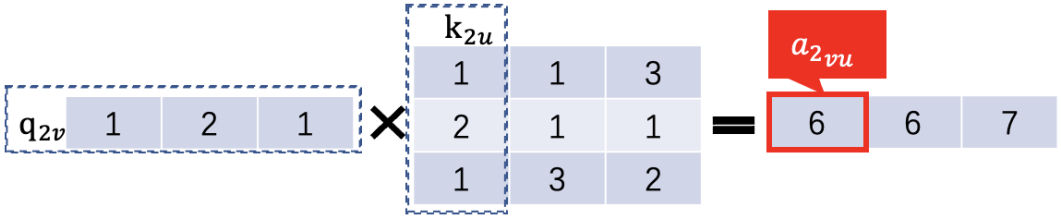

Fig. 5.7. The example of the dot-production attention function.

where $W_{att1} \in \mathbb{R}^{d_{in} \times d_{out}}$ and $W_{att2} \in \mathbb{R}^{d_{out} \times 1}$ are learnable weights, d_{in} and d_{out} are the dimensions of the W_{att1}. Then, we normalize the $qk_{vu}^{''}$ as the same as the Transformer [51] and utilize SoftMax to calculate the attention weights between the target node v and the neighboring node u as follows:

$$att_{vu} = \text{SoftMax}(\frac{\text{qk}_{vu}^{''}}{\sqrt{d_{out}}}), u \in N(v) \tag{5.21}$$

Weighted sum aggregation

Before we use attention weights to aggregate the neighborhood of target node v, we first do a feature transformation for the input,

$$\text{v}_u = W_v \cdot h_u, u \in N(v), \tag{5.22}$$

where W_v are learnable weighta. Then we aggregate the neighborhood of the target node v, notably, we found that position embedding and structure embedding are important for both the attention generation and the feature transformation. Therefore, our weighted sum aggregation is calculated as follows:

$$z_v = \sum_{u \in N(v)} att_{vu} \cdot (\text{v}_u + \text{se}_u + p_u). \tag{5.23}$$

5.3.5 Integration

The input to the integration is the concatenation of the node-wise features of target node v and the output of the weighted sum aggregation. Therefore, our integration is summarized as follows:

$$h_v' = \sigma\left(W_l \cdot \text{cat}\left(h_v, z_v\right)\right). \tag{5.24}$$

where W_l is learnable weight. Hereafter, we represent the set of these operations as $h_V^{'} :=$ MTSPAGC(h_V).

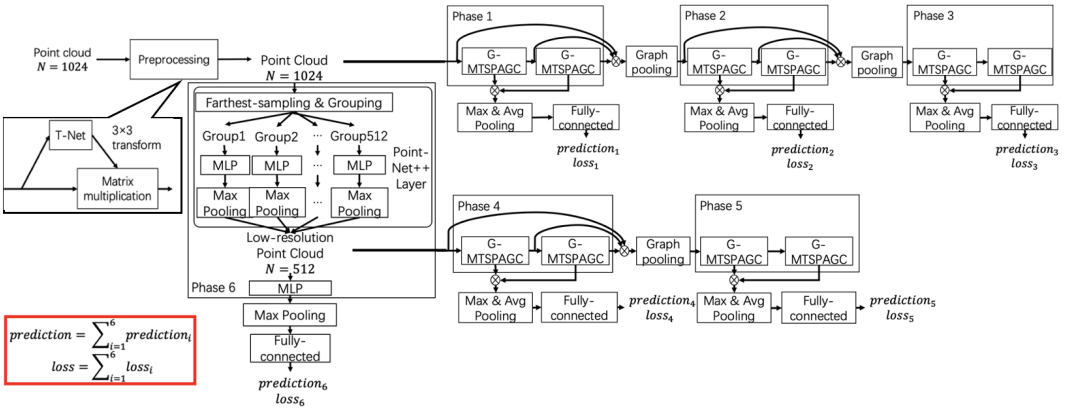

Fig. 5.8. Architecture of the 3D point cloud classification network where $\otimes$ represents the concatenation operation. The model adds up the losses and predictions from the different phases to obtain the overall classification loss and final prediction.

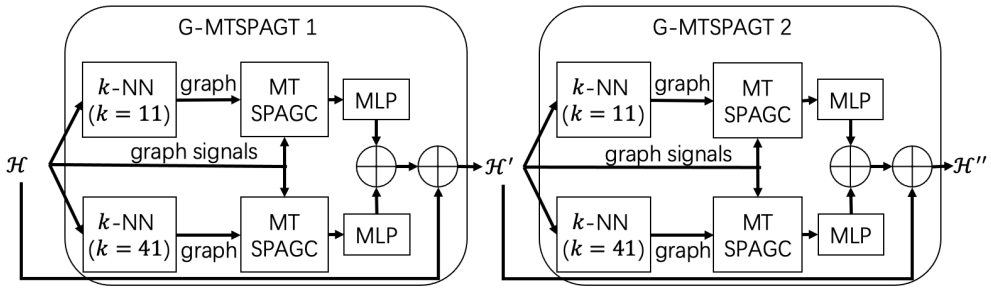

Fig. 5.9. The structure of multi-head MTSPAGC module. We have two modules (G-MTSPAGC 1 and G-MTSPAGC 2), and two k-NN are applied to the input to create dynamic edges in each head.

5.4 3D Point Clouds Classification Network

In this section, we introduce a 3D point cloud classification network using the MTSPAGC. Fig. 5.8 shows the overall architecture of the MTSPAGC-based 3D point cloud classification network. We represent the input 3D point cloud as $X = \{x_i\}_{i=1}^{N}$, where N is the number of points.

5.4.1 Description

We preprocess the point cloud using the same transformation module (T-Net) as PointNet [68], which os introduced in section 3.4.1, to lessen rotational effects. In addition, we build a low-resolution point cloud using a layer of PointNet++ [65], which is introduced in section 3.4.1. The point cloud may simply be used to extract both global and local information because of the multi-resolution structure.

The specifics of the building blocks created expressly for point cloud classification are described in the follow sections.

5.4.2 Multi-head MTSPAGC module

The structure of the multi-head MTSPAGC module (G-MTSPAGC in Fig. 5.8) is shown in Fig. 5.9. The input of this module is a set of F-dimensional feature from the previous layer.

$\mathcal{H} = \{h_j\}_{j=1}^{M}$ is the input of this module, with M denoting the number of features in the input. For the first module, $\mathcal{H}$ is only X and $F = 3$, which stand for the x, y, and z

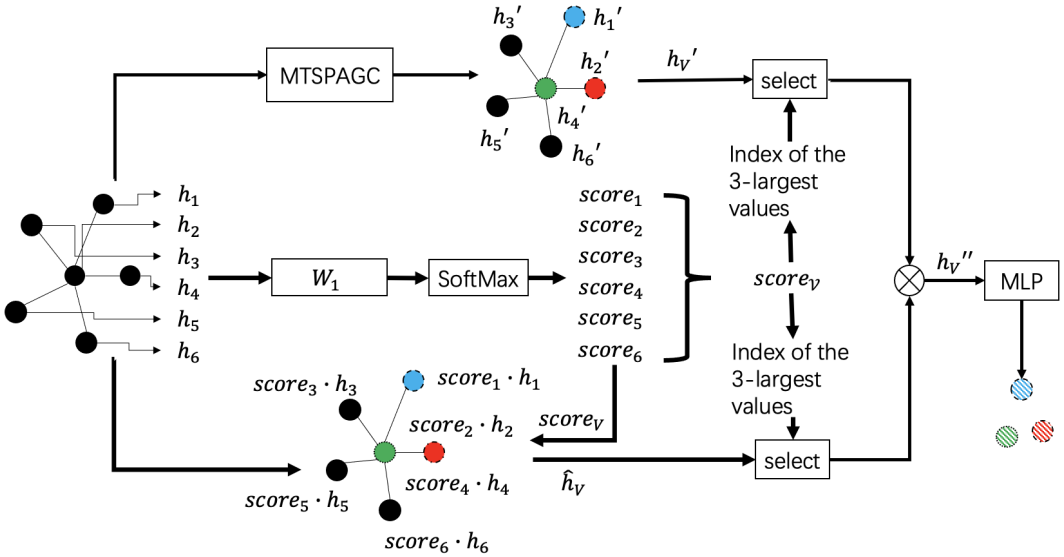

Fig. 5.10. The example of our graph pooling for $w = 3$, where $\otimes$ is the concatenation operation.

coordinates of the points.

We use k-NN constructing a graph for $\mathcal{H}$. Considering the value of k may effect the performance of the module, for each head, we use different $k = \{11, 15\}$ to construct graph. Simultaneously, multiple graph signals are processed using two-haed MTSPAGC inspired by ResNeXt [69].

Dynamic graph convolution, which allows the graph topology to change at each layer, outperforms a fixed graph structure, according to [58, 70]. As a result, different graphs are created for different G-MTSPAGC modules.

5.4.3 Graph Pooling

A hot topic in GNNs and graph signal processing is efficient graph pooling methods [53, 54]. Earlier studies used graph coarsening and global node representation pooling techniques.

Two trainable graph pooling techniques, DiffPool [59] and GraphU-net [60], were recently proposed. We provide a score-based graph pooling method that draws inspiration from GraphU-net.

Fig. 5.10 shows our graph pooling. In our graph pooling, it has two branches. The first branch is named local branch, and the second branch is named global score branch. Finally, the outputs of these two branches are fused by an MLP.

Local branch

In this branch, we use our MTSPAGC processing the input. Therefore, each proposed node conclude the local information. This process is represented as follows:

$$h'_V = \text{MTSPAGC}(h_V). \tag{5.25}$$

Global score branch

In our global score branch, we first embed node-wise features as follows:

$$\hat{h}_v = \sigma(W_{\text{p}} \cdot h_v), v \in V, \tag{5.26}$$

where W_{p} is the shared learnable weight.

Then, we calculate a global score for each node as follows:

$$\begin{aligned} \mathrm{score}_v &= \mathrm{SoftMax}(W_1 \cdot \hat{h}_v) \\ &:= \frac{\exp\left(W_1 \cdot \hat{h}_v\right)}{\sum_{u \in V} \exp\left(W_1 \cdot \hat{h}_u\right)}, v \in V, \end{aligned} \tag{5.27}$$

where W_1 is the shared learnable weight. The scores are then used to update the features on the nodes, i.e., $\hat{h}_V = \{\mathrm{score}_v \cdot \hat{h}_v\}$.

Finally, for each branch, we place the nodes in descending order based on their scores and choose the top w nodes. For the global score branch,

$$h_{\mathrm{select}\,g} = \hat{h}_V[\mathrm{idx}_{\mathrm{select}}], \tag{5.28}$$

For local branch,

$$h_{\mathrm{select}\,l} = h'_V[\mathrm{idx}_{\mathrm{select}}], \tag{5.29}$$

where $h_{\mathrm{select}\,g}$ and $h_{\mathrm{select}\,l}$ are the set of node-wise features in the smaller graph. $\mathrm{idx}_{\mathrm{select}} = \mathrm{rank}\left(\{\mathrm{score}_v\}_V, w\right)$ is the indices of the w nodes selected, in which $\mathrm{rank}\left(\cdot\right)$ is the node ranking operation that returns indices of the w highest scores.

As shown in Fig. 5.10, we select $w = 3$ nodes from the learned scores.

Feature fusion

In our graph pooling, we use a MLP to fuse the outputs of each branch. We first concatenate the output of the local branch and the output of the global score branch. Then, we input it into a MLP layer to get the final output. The feature fusion can be represented as follows:

$$\begin{aligned} h_{f_V} &:= \mathrm{concatenate}(h_{\mathrm{select}\,g}, h_{\mathrm{select}\,l}), \\ h_{f_V} &= \mathrm{MLP}(h_{f_V}). \end{aligned} \tag{5.30}$$

According to the process, we can know that unlike GraphU-net, which outputs the picked w nodes directly, our graph pooling also process the features of the selected nodes using the MTSPAGC. Therefore, the output nodes can contain more information.

5.4.4 Hierarchical Prediction Architecture

We use the inter-mediate supervision technique [71] to fully use the hierarchical features and propose a hierarchical prediction architecture (Fig. 5.8). In conclusion, we take into account the loss at each G-MTSPAGC module when calculating the overall loss function.

Two G-MTSPAGC modules are utilized in each phase. We invested maximum and average pooling layers there. The input to a fully connected layer is then created by concatenating these outputs. We calculate the classification loss and the prediction label for each stage. The total classification loss as well as the final prediction labels include the losses from various phases. The following illustration illustrates this processing:

$$prediction = \sum_{i=1}^{P} prediction_i, \tag{5.31}$$

$$loss = \sum_{i=1}^{P} loss_i, \tag{5.32}$$

where *prediction* is the final prediction value, $loss_i$ is the cross entropy loss. *loss* is the overall classification loss, and P is the number of phases.

This architecture has the benefit of allowing us to combine the outcomes of the several phases to produce predictions that are more dependable and robust.

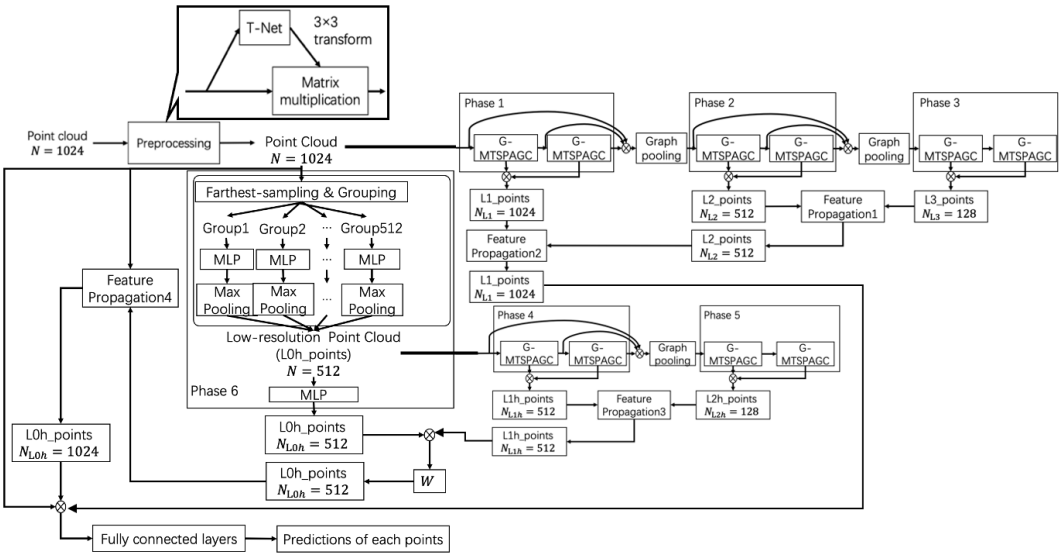

Fig. 5.11. Architecture of the 3D point cloud part segmenation network where $\otimes$ represents the concatenation operation. N_{L1} is the number of points of L1_points. N_{L2} is the number of points of L2_points. N_{L3} is the number of points of L3_points. N_{L0h} is the number of points of L0h_points. N_{L1h} is the number of points of L1h_points. N_{L2h} is the number of points of L2h_points. W is learnable weights.

5.5 3D Point Cloud Part Segmentation Network

In this section, we introduce a 3D point cloud part segmentation network using the MTSPAGC. Fig. 5.11 shows the overall architecture of the MTSPAGC-based 3D point cloud part segmantation network. We represent the input 3D point cloud as $X = \{x_i\}_{i=1}^{N}$, where N is the number of points.

5.5.1 Description

As shown in Fig. 5.11, our 3D point cloud part segmentation network is based on our 3D point cloud classification network. Therefore, the G-MTSPAGC modules are the same as described in section 5.4.2. The graph pooling layers are the same as described in section 5.4.3. The PointNet++ layer and preprocess are also the same as our 3D point cloud classification network. In the following, we introduce the feature propagation layers.

5.5.2 Feature Propagation

Graph pooling can be seen as a down-sampling operation. In the counterpart, feature propagation can be seen as an up-sampling operation. Feature Propagation layers utilized in our 3D point cloud part segmentation network are the same as that utilized in PointNet++ [65]. Here, we let the point cloud before a graph pooling layer is $P_1 = \{p_{1i}\}_{i=1}^{N_1}$, where the number of points in P_1 is N_1. We let the point cloud after a graph pooling layer is $P_2 = \{p_{2j}\}_{j=1}^{N_2}$, where the number of points in P_2 is N_2. Also, $N_2 < N_1$.

First, we calculate distance between a point in P_1 and points in P_2. This can be shown as follows:

$$dis_i := \{\|p_{1i}, p_{2j}\|_2\}_{j=1}^{N_2}, p_{1i} \in P_1, p_{2j} \in P_2, \tag{5.33}$$

where, $\|\cdot\|_2$ is Euclidean distance.

We find m nearest points in P_2 to point p_{1i} in P_1 and the corresponding distance. They can be summarized as follows:

$$\begin{aligned} subP_2 &:= \{p_{2j}\}_{j=1}^{m}, \\ subdis_i &:= \{dis_{ij}\}_{j=1}^{m}. \end{aligned} \tag{5.34}$$

Then, we can calculate a weight between p_{1i} and a point in $subP_2$ as follows:

$$\begin{aligned} norm &= \sum_{j=1}^{m} \frac{1}{dis_{ij}}, dis_{ij} \in subdis_i, \\ W_{FPij} &= \frac{1}{dis_{ij} \cdot norm}, dis_{ij} \in subdis_i. \end{aligned} \tag{5.35}$$

Then, weighted sum is utilized to aggregate the points in $subP_2$ as follows:

$$p_{ti} = \sum_{j=1}^{m} W_{FPij} \cdot p_{2j}, p_{2j} \in subP_2. \tag{5.36}$$

Finally, the output of feature propagation can be calculated as follows:

$$P_1{}' = \{\mathrm{MLP}(\mathrm{cat}(p_{1i}, p_{ti}))\}_{i=1}^{N_1}, \tag{5.37}$$

where, $\mathrm{cat}(\cdot)$ is the concatenation operation, $\mathrm{MLP}(\cdot)$ is a multilayer perceptron.

5.6 Experimental Results of point clouds classification

In this section, we introduce our experiments for point cloud classification. We tested performance of the proposed GNN using the ModelNet dataset [66].

5.6.1 Dataset introduce

A total of 12,308 computer-aided design (CAD) models in 40 categories are available on ModelNet40, including 2,468 for testing and 9,840 for training. 4,899 CAD models total, 3,991 for training and 908 for testing, are included in ModelNet10. We used the same data structure as earlier 3D point cloud classification methods: 1,024 points were evenly sampled among the faces of the CAD mesh. All point clouds were initially adjusted to be contained within a unit sphere. The example of ModelNet is shown in Fig. 5.12.

5.6.2 Settings and evaluates

Table 5.1 displays the settings of hyper parameters of the point cloud classification network.

We evaluated the accuracy of classification with other networks created specifically for point cloud classification. Due to the class imbalance in the dataset, we employed two performance metrics to evaluate the outcomes: average accuracy of all test cases (OA) and average accuracy of all shape classes (mAcc).

5.6.3 Results and explanation

Table 5.2 provides a summary of the results, with scores of the alternative approaches culled from the related sources. We discovered that our network had higher accuracy than the other methods on both OA and mAcc. We believe that graph-based approaches can provide greater relationship information between points than the Pointwise MLP methods and the Convolution-based methods. Additionally, when compared to existing graph-based methods,

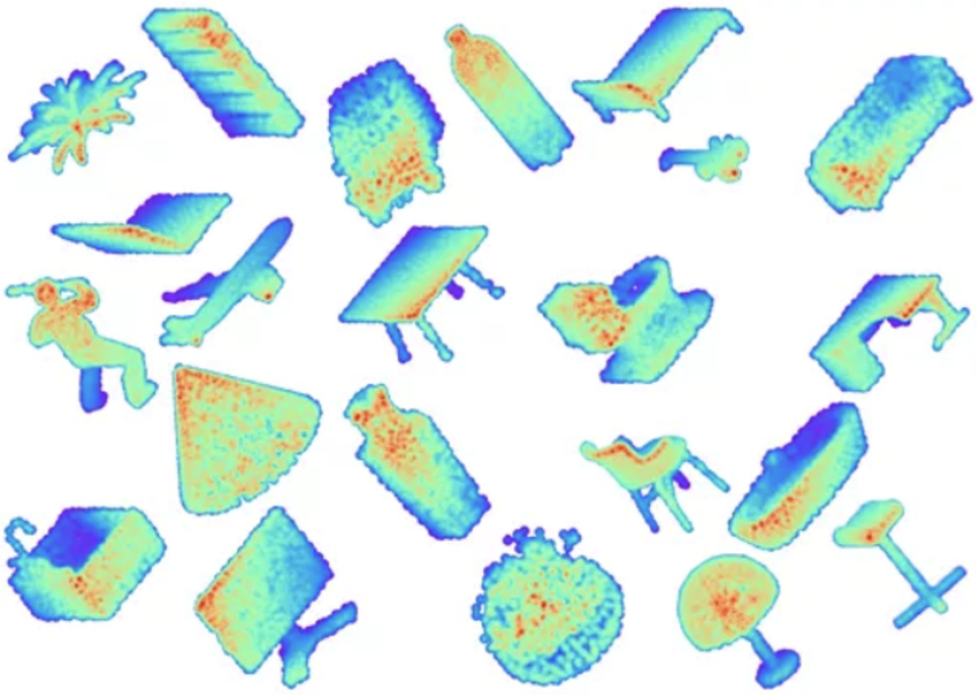

Fig. 5.12. The example of ModelNet dataset [66].

our method uses proposed structural features to obtain detailed local structural information in feature space in addition to relational information between points. Finally, by using a multi-resolution point cloud, our method can concurrently use local and global information from the point cloud. Therefore, our method can had higher accuracies than the other approaches.

In the Table 5.3 we compared our methods with other transformer-based methods. We also can see that our method had higher accuracy. First, we think our method can get a better attention weights by using our multi-type structure-position-aware self-attention mechanism. Therefore, our method can utilizes information in neighborhood of target node better.

5.7 Experimental Results of point clouds part segmentation

In this section, we introduce our experiments for point cloud part segmentation. We tested performance of the proposed GNN using the ShapeNet dataset [90].

5.7.1 Dataset introduce

The ShapeNet dataset [90] is a richly labeled, large-scale 3D point clouds dataset. The ShapeNet has 16 classes, 50 parts and a total of 16,872 point clouds. The samples in this dataset are uneven, e.g. Table contains 5263 point clouds, while Earphone has only 69 point clouds. Each point cloud contains 2,048 points. ShapeNet includes 2,874 point clouds for testing and 13,998 point clouds for training. The examples of ShapeNet is shown in Fig. 5.13.

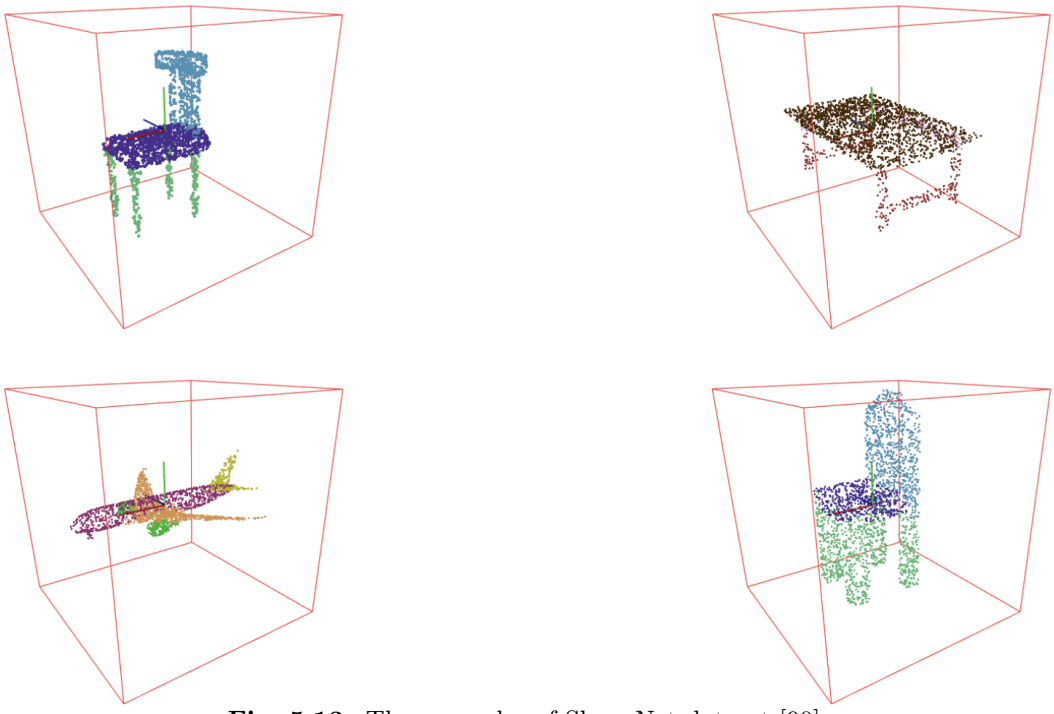

Fig. 5.13. The examples of ShapeNet dataset [90].

5.7.2 Settings and evaluates

Table 5.4 displays the settings of hyper parameters of the point cloud part segmentation network.

We evaluated the average mIoU of all test instance with other networks created specifically for point cloud part segmentation. We also show the mIoU of each class.

5.7.3 Results and explanation

Table 5.5 provides a summary of the results, where the scores of the alternative approaches are obtained from experiments using the same settings shown in Table 5.4. We found that our network had the same average mIoU of all test instances as the graph-based methods, i.e., DGCNN [70], and outperformed other existing methods.

Compared with other methods, we think our method can obtain detailed local structural information in the feature space. In addition, our method can utilize information of neighborhood more efficiently by introducing attention mechanism. Furthermore, by using multiple type attention function simultaneously, our method can get better attention weights. These reasons lead to our method having a good performance.

5.8 Effect of MTSPAGC

In this section, we evaluate the effectiveness of each part of our method. We performed experiments on a graph classification problem for point clouds using different transformer layers, which are introduced as follows. Here, we utilize a simplified network of our point cloud classification network.

Table 5.1: The hyper parameters of the point cloud classification network. k is the value of k-NN, w is the number of selected nodes. We set t, the number of heads, for the multi-head MTSPAGC (G-MTSPAGC) introduced in Section 5.4.2. For Pointnet++ layer, S is the number of sampled points, r is the radius of each group, D is the number of points of each group.

Learning rate	0.001335			
Drop out	0.3			
t	2			
Graph pooling layername	k	w	[input channels, output channels]	
Graph pooling 0	15	512	[64,64]	
Graph pooling 1	15	128	[128,128]	
Graph pooling 2	15	128	[128,128]	
Graph convolution layername	k	[input channels, output channels]		
G-MTSPAGC 0	[11,15]	[3,64]		
G-MTSPAGC 1	[11,15]	[64,64]		
G-MTSPAGC 2	[11,15]	[64,64]		
G-MTSPAGC 3	[11,15]	[64,128]		
G-MTSPAGC 4	[11,15]	[128,256]		
G-MTSPAGC 5	[11,15]	[256,256]		
G-MTSPAGC 6	[11,15]	[128,128]		
G-MTSPAGC 7	[11,15]	[128,128]		
G-MTSPAGC 8	[11,15]	[128,256]		
G-MTSPAGC 9	[11,15]	[256,256]		
Pointnet++ layer name	S	r	D	[input channels, output channels]
Pointnet++	512	0.2	32	[3,64]
		0.4	128	[3,64]

5.8.1 Subtraction attention-based graph convolution (SUBGC)

In this module, we only use subtraction as the attention function without position embedding and structure embedding. The architecture of the simplified network is shown in Fig. 5.14. Each G-SUBGC module has two heads, therefore, the k of k-NN is also $k = \{11, 15\}$. The SUBGC has two parts:

1) Subtraction weighted sum aggregation;

2) Integration.

Subtraction weighted sum aggregation The subtraction attention weights are calculated as follows:

$$\begin{aligned} a_{vu} = f(\mathrm{q}_v, \mathrm{k}_u) &:= \{\mathrm{q}_v - \mathrm{k}_u\}_{u \in N(v)}, \\ \mathrm{qk}_{vu} &= W_{att2} \cdot (\sigma(W_{att1} \cdot (a_{vu}))), u \in N(v) \\ att_{vu} &= \mathrm{SoftMax}(\frac{\mathrm{qk}_{vu}}{\sqrt{d_{out}}}), u \in N(v) \end{aligned} \tag{5.38}$$

where, $\mathrm{q}_v = W_q \cdot h_v$, $\mathrm{k}_u = W_k \cdot h_u, u \in N(v)$, and $\mathrm{v}_u = W_v \cdot h_u, u \in N(v)$, W_q, W_k and W_v are learnable weights. $W_{att1} \in \mathbb{R}^{d_{in} \times d_{out}}$ and $W_{att2} \in \mathbb{R}^{d_{out} \times 1}$ are learnable weights, d_{in} and d_{out} are the dimensions of the W_{att1}.

The weighted sum aggregation is

$$z_v = \sum_{u \in N(v)} att_{vu} \cdot (\mathrm{v}_u). \tag{5.39}$$

Table 5.2: Comparison results of the 3D shape classification on the ModelNet benchmark. OA indicates the average accuracy of all test instances, and mAcc indicates the average accuracy of all shape categories. The symbol "-" indicates that the results are not available from the references.

Type	Method	ModelNet40		ModelNet10	
		OA	mAcc	OA	mAcc
Pointwise MLP Methods	PointNet [68]	89.2%	86.2%	-	-
	PointNet++ [65]	90.7%	-	-	-
	SRN-PointNet++ [82]	91.5%	-	-	-
	PointASNL [83]	93.2%	-	95.9%	-
Convolution-based Methods	PointConv [84]	92.5%	-	-	-
	A-CNN [85]	92.6%	90.3%	95.5%	95.3%
	SFCNN [86]	92.3%	-	-	-
	InterpCNN [87]	93.0%	-	-	-
	ConvPoint [88]	91.8%	88.5%	-	-
Graph-based Methods	ECC [58]	87.4%	83.2%	90.8%	90.0%
	DGCNN [70]	92.2%	90.2%	-	-
	LDGCNN [78]	92.9%	90.3%	-	-
	Hassani et al. [79]	89.1%	-	-	-
	DPAM [72]	91.9%	89.9%	94.6%	94.3%
	KCNet [77]	91.0%	-	94.4%	-
	ClusterNet [80]	87.1%	-	-	-
	RGCNN [73]	90.5%	87.3%	-	-
	LocalSpecGCN [74]	92.1%	-	-	-
	PointGCN [75]	89.5%	86.1%	91.9%	91.6%
	3DTI-Net [76]	91.7%	-	-	-
	Grid-GCN [81]	93.1%	91.3%	97.5%	97.4%
	SAGC	93.5%	91.3%	98.3%	97.7%
	SAMGC	93.6%	91.4%	98.3%	**97.8%**
	MTSPAGC	**93.8%**	**91.5%**	**98.4%**	**97.8%**

Integration The integration is defined as follows:

$$h'_v = \sigma\left(W_l \cdot \mathrm{cat}(h_v, z_v)\right). \tag{5.40}$$

where W_l is learnable weight.

The settings of networks is summarized in Table 5.6. The graph pooling layer is introduced section 5.4.3.

5.8.2 Dot-production attention-based graph convolution (DotGC)

In this module, we use dot-production as the attention function without position embedding and structure embedding. The architecture of the simplified network is shown in Fig. 5.15. Each G-DotGC module has two heads, therefore, the k of k-NN is also $k = \{11, 15\}$. The DotGC has two parts:

1) Dot-production weighted sum aggregation;

2) Integration.

Dot-production weighted sum aggregation The dot-production attention weights

Table 5.3: Comparison results of the 3D shape classification on the ModelNet benchmark. OA indicates the average accuracy of all test instances.

Type	Method	ModelNet40
		OA
Transformer-based Methods	A-ShapeContextNet [95]	90.0%
	Yang et al. [96]	91.7%
	3DETR [97]	91.9%
	Point Transformer [98]	92.8%
	MLMSPT [99]	92.9%
	DTNet [100]	92.9%
	PointASNL [83]	93.2%
	Wu et al. [101]	93.2%
	LFT-Net [102]	93.2%
	PCT [103]	93.2%
	3DCTN [104]	93.3%
	3DMedPT [105]	93.4%
	3CROSSNet [106]	93.5%
	PAT [107]	93.6%
	PointTransformer [94]	93.7%
	MTSPAGC	**93.8%**

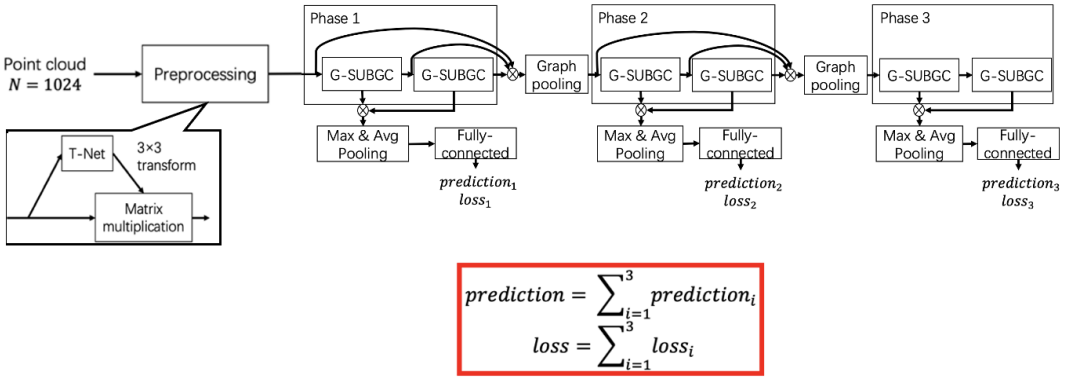

Fig. 5.14. The architecture of the simplified network, which utilize SUBGC.

are calculated as follows:

$$\begin{aligned} a_{vu} = f(\mathrm{q}_v, \mathrm{k}_u) &:= \{\mathrm{q}_v \cdot \mathrm{k}_u^{\mathrm{T}}\}_{u \in N(v)}, \\ \mathrm{qk}_{vu} &= a_{vu}, u \in N(v) \\ att_{vu} &= \mathrm{SoftMax}(\frac{\mathrm{qk}_{vu}}{\sqrt{d_k}}), u \in N(v) \end{aligned} \tag{5.41}$$

where, $\mathrm{q}_v = W_q \cdot h_v$, $\mathrm{k}_u = W_k \cdot h_u, u \in N(v)$, and $\mathrm{v}_u = W_v \cdot h_u, u \in N(v)$, W_q, W_k and W_v are learnable weights. d_k is the dimensions of the k_u.

The weighted sum aggregation is

$$z_v = \sum_{u \in N(v)} att_{vu} \cdot (\mathrm{v}_u) \tag{5.42}$$

Integration The Integration is defined as follows:

$$h'_v = \sigma\left(W_l \cdot \mathrm{cat}(h_v, z_v)\right). \tag{5.43}$$

Table 5.4: The hyper parameters of the point cloud part segmentation network. k is the value of k-NN, w is the number of selected nodes. We set t, the number of heads, for the multi-head MTSPAGC (G-MTSPAGC) introduced in Section 5.4.2. For Pointnet++ layer, S is the number of sampled points, r is the radius of each group, D is the number of points of each group.

Learning rate	0.001335			
Drop out	0.3			
t	2			
Graph pooling layername	k	w	[input channels, output channels]	
Graph pooling 0	15	512	[64,64]	
Graph pooling 1	15	128	[128,128]	
Graph pooling 2	15	128	[128,128]	
Graph convolution layername	k	[input channels, output channels]		
G-MTSPAGC 0	[11,15]	[3,64]		
G-MTSPAGC 1	[11,15]	[64,64]		
G-MTSPAGC 2	[11,15]	[64,64]		
G-MTSPAGC 3	[11,15]	[64,128]		
G-MTSPAGC 4	[11,15]	[128,256]		
G-MTSPAGC 5	[11,15]	[256,256]		
G-MTSPAGC 6	[11,15]	[128,128]		
G-MTSPAGC 7	[11,15]	[128,128]		
G-MTSPAGC 8	[11,15]	[128,256]		
G-MTSPAGC 9	[11,15]	[256,256]		
Pointnet++ layer name	S	r	D	[input channels, output channels]
Pointnet++	512	0.2	32	[3,64]
		0.4	128	[3,64]
Feature propagation layername	channels of MLP			
Feature Propagation1	[256,128]			
Feature Propagation2	[128,128]			
Feature Propagation3	[256,128]			
Feature Propagation4	[128,128]			

where W_l is learnable weight.

The settings of networks is summarized in Table 5.7. The graph pooling layer is introduced section 5.4.3.

5.8.3 Multi-type attention-based graph convolution (MTGC)

In this module, we use a multi-type attention function, which is introduced in section 5.3.4, without position embedding and structure embedding. The architecture of the simplified network is shown in Fig. 5.16. Each G-MTGC module has two heads, therefore, the k of k-NN is also $k = \{11, 15\}$. The MTGT has two parts:

1) Multi-type weighted sum aggregation;

2) Integration.

Multi-type weighted sum aggregation The multi-type attention weights are calcu-

Table 5.5: Comparison results of the 3D point cloud part segmentation on the ShapeNet benchmark. mIoU indicates the average mIoU of all test instances. The mIoU of each class is also shown.

Method	mIoU	air-plane	bag	cap	car	chair	ear-phone	guitar	knife	lamp	lap-top	motor	mug	pistol	rocket	skate-board	table
PointNet [68]	83.7	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
Point-Net++ [65]	**85.1**	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
DGCNN [70]	**85.1**	**82.7**	79.9	**84.4**	77.1	90.5	77.8	91.0	87.0	84.2	**95.7**	61.9	93.0	79.9	58.2	74.2	83.2
Point-ASNL [83]	84.6	82.4	**80.3**	83.2	76.8	89.9	**80.6**	90.8	86.7	84.2	95.6	60.1	**94.6**	**81.6**	**59.1**	73.7	82.3
MTSPAGC	**85.1**	81.6	76.1	83.4	**78.2**	90.5	72.3	90.9	**87.5**	**84.3**	95.4	65.9	94.3	81.3	55.4	74.5	**83.5**

Table 5.6: The settings of the simplified point cloud classification network using SUBGC. k is the value of k-NN, w is the number of selected nodes. We set t, the number of heads, for the multi-head graph convolution introduced before.

Learning rate	0.00135		
Drop out	0.3		
t	2		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
SUBGC 0	[11,15]	[3,64]	
SUBGC 1	[11,15]	[64,64]	
SUBGC 2	[11,15]	[64,64]	
SUBGC 3	[11,15]	[64,128]	
SUBGC 4	[11,15]	[128,256]	
SUBGC 5	[11,15]	[256,256]	

lated as follows:

$$\begin{aligned} a_{1vu} &= f_1(\mathrm{q}_v, \mathrm{k}_u) := \{\mathrm{q}_v - \mathrm{k}_u\}_{u\in N(v)}, \\ a_{2vu} &= f_2(\mathrm{q}_v, \mathrm{k}_u) := \{\mathrm{q}_v \cdot \mathrm{k}_u^{\mathrm{T}}\}_{u\in N(v)}, \\ \mathrm{qk}_{vu} &= a_{1vu} + W_c \cdot a_{2vu}, u \in N(v), \\ \mathrm{qk}_{vu} &= W_{att2} \cdot (\sigma(W_{att1} \cdot (\mathrm{qk}_{vu}))), u \in N(v) \\ att_{vu} &= \mathrm{SoftMax}(\frac{\mathrm{qk}_{vu}}{\sqrt{d_{out}}}), u \in N(v) \end{aligned} \tag{5.44}$$

where, $\mathrm{q}_v = W_q \cdot h_v$, $\mathrm{k}_u = W_k \cdot h_u, u \in N(v)$, and $\mathrm{v}_u = W_v \cdot h_u, u \in N(v)$, W_q, W_k,W_v and W_c are learnable weights. $W_{att1} \in \mathbb{R}^{d_{in}\times d_{out}}$ and $W_{att2} \in \mathbb{R}^{d_{out}\times 1}$ are learnable weights, d_{in} and d_{out} are the dimensions of the W_{att1}.

The weighted sum aggregation is

$$z_v = \sum_{u\in N(v)} att_{vu} \cdot (\mathrm{v}_u). \tag{5.45}$$

Integration The integration is defined as follows:

$$h'_v = \sigma\left(W_l \cdot \mathrm{cat}(h_v, z_v)\right), \tag{5.46}$$

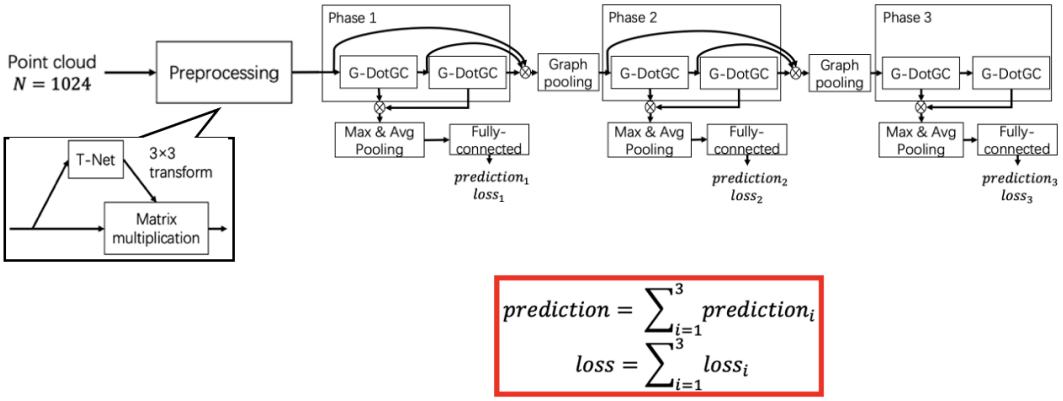

Fig. 5.15. The architecture of the simplified network, which utilize DotGC.

Table 5.7: The settings of the simplified point cloud classification network using DotGC. k is the value of k-NN, w is the number of selected nodes. We set t, the number of heads, for the multi-head graph convolution introduced before.

Learning rate	0.00135		
Drop out	0.3		
t	2		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
DotGC 0	[11,15]	[3,64]	
DotGC 1	[11,15]	[64,64]	
DotGC 2	[11,15]	[64,64]	
DotGC 3	[11,15]	[64,128]	
DotGC 4	[11,15]	[128,256]	
DotGC 5	[11,15]	[256,256]	

where W_l is learnable weight.

The settings of networks is summarized in Table 5.8. The graph pooling layer is introduced section 5.4.3.

5.8.4 Multi-type position-aware attention-based graph convolution (MTPAGC)

In this module, we use a multi-type attention function and position embedding, which are introduced in section 5.3.4 and section 5.3.3, without structure embedding. The architecture of the simplified network is shown in Fig. 5.17. Each G-MTPAGC module has two heads, therefore, the k of k-NN is also $k = \{11, 15\}$. The MTPAGC has two parts:

1) Multi-type position-aware weighted sum aggregation;

2) Integration.

Multi-type position-aware weighted sum aggregation The multi-type position-

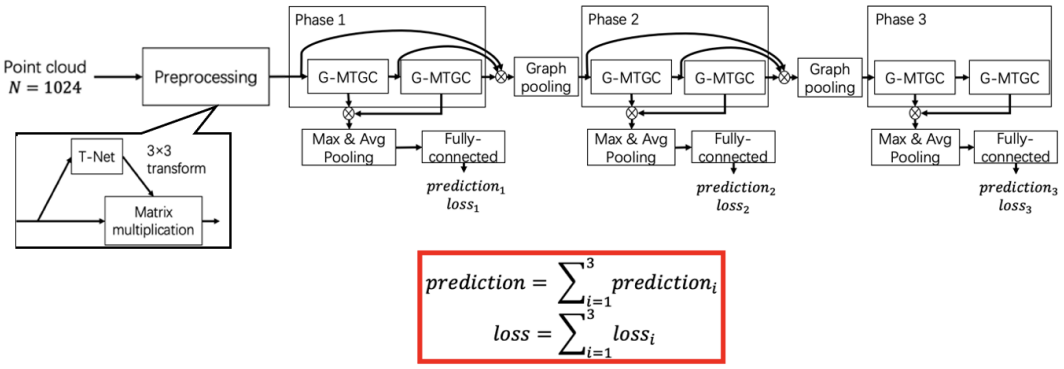

Fig. 5.16. The architecture of the simplified network, which utilize MTGC.

Table 5.8: The settings of the simplified point cloud classification network using MTGC. k is the value of k-NN, w is the number of selected nodes. We set t, the number of heads, for the multi-head graph convolution introduced before.

Learning rate	0.00135		
Drop out	0.3		
t	2		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
MTGC 0	[11,15]	[3,64]	
MTGC 1	[11,15]	[64,64]	
MTGC 2	[11,15]	[64,64]	
MTGC 3	[11,15]	[64,128]	
MTGC 4	[11,15]	[128,256]	
MTGC 5	[11,15]	[256,256]	

aware attention weights are calculated as follows:

$$\begin{aligned} a_{1vu} &= f_1(\mathrm{q}_v, \mathrm{k}_u) := \{\mathrm{q}_v - \mathrm{k}_u\}_{u\in N(v)}, \\ a_{2vu} &= f_2(\mathrm{q}_v, \mathrm{k}_u) := \{\mathrm{q}_v \cdot \mathrm{k}_u^{\mathrm{T}}\}_{u\in N(v)}, \\ \mathrm{qk}_{vu} &= a_{1vu} + W_c \cdot a_{2vu}, u \in N(v), \\ \mathrm{qk}_{vu} &= W_{att2} \cdot (\sigma(W_{att1} \cdot (\mathrm{qk}_{vu} + p_u))), u \in N(v) \\ att_{vu} &= \mathrm{SoftMax}(\frac{\mathrm{qk}_{vu}}{\sqrt{d_{out}}}), u \in N(v) \end{aligned} \tag{5.47}$$

where, p_u is the position embedding which is introduced in section 5.3.3, $\mathrm{q}_v = W_q \cdot h_v$, $\mathrm{k}_u = W_k \cdot h_u, u \in N(v)$, and $\mathrm{v}_u = W_v \cdot h_u, u \in N(v)$, W_q, W_k,W_v and W_c are learnable weights. $W_{att1} \in \mathbb{R}^{d_{in}\times d_{out}}$ and $W_{att2} \in \mathbb{R}^{d_{out}\times 1}$ are learnable weights, d_{in} and d_{out} are the dimensions of the W_{att1}.

The weighted sum aggregation is

$$z_v = \sum_{u\in N(v)} att_{vu} \cdot (\mathrm{v}_u + p_u). \tag{5.48}$$

Integration The integration is defined as follows:

$$h'_v = \sigma\left(W_l \cdot \mathrm{cat}(h_v, z_v)\right). \tag{5.49}$$

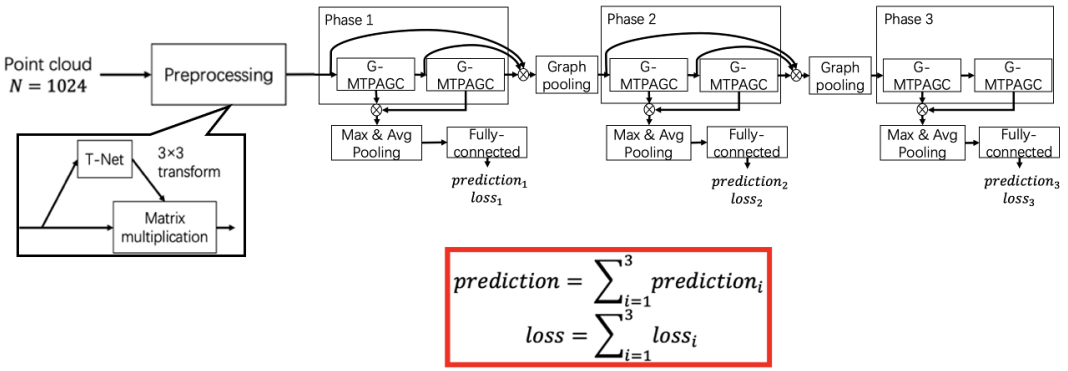

Fig. 5.17. The architecture of the simplified network, which utilize MTPAGC.

Table 5.9: The settings of the simplified point cloud classification network using MTPAGC. k is the value of k-NN, w is the number of selected nodes. We set t, the number of heads, for the multi-head graph convolution introduced before.

Learning rate	0.00135		
Drop out	0.3		
t	2		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
MTPAGC 0	[11,15]	[3,64]	
MTPAGC 1	[11,15]	[64,64]	
MTPAGC 2	[11,15]	[64,64]	
MTPAGC 3	[11,15]	[64,128]	
MTPAGC 4	[11,15]	[128,256]	
MTPAGC 5	[11,15]	[256,256]	

where W_l is learnable weight.

The settings of networks is summarized in Table 5.9. The graph pooling layer is introduced section 5.4.3.

5.8.5 Multi-type structure and position-aware attention-based graph convolution (MTSPAGC)

In this module, we use the completed multi-type structure and position-aware attention-based graph convolution, which are introduced in section 5.3. The architecture of the simplified network is shown in Fig. 5.18. Each G-MTSPAGC module has two heads, therefore, the k of k-NN is also $k = \{11, 15\}$. The MTSPAGC has two parts:

1) Multi-type structure-position-aware weighted sum aggregation;

2) Integration.

Multi-type structure and position-aware weighted sum aggregation The multi-type structure and position-aware attention weights are introduced in section 5.3. It calcu-

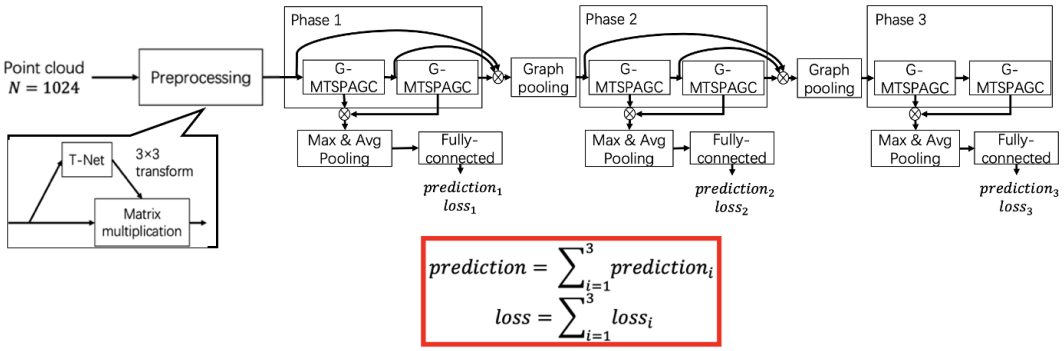

Fig. 5.18. The architecture of the simplified network, which utilize MTSPAGC.

lated as follows:

$$\begin{aligned} a_{1vu} &= f_1(\mathrm{q}_v, \mathrm{k}_u) := \{\mathrm{q}_v - \mathrm{k}_u\}_{u \in N(v)}, \\ a_{2vu} &= f_2(1_v, \mathrm{k}_u) := \{\mathrm{q}_v \cdot \mathrm{k}_u^{\mathrm{T}}\}_{u \in N(v)}, \\ \mathrm{qk}_{vu} &= a_{1vu} + W_c \cdot a_{2vu}, u \in N(v), \\ \mathrm{qk}_{vu} &= W_{att2} \cdot (\sigma(W_{att1} \cdot (\mathrm{qk}_{vu} + \mathrm{se}_u + p_u))), u \in N(v) \\ att_{vu} &= \mathrm{SoftMax}(\frac{\mathrm{qk}_{vu}}{\sqrt{d_{out}}}), u \in N(v) \end{aligned} \tag{5.50}$$

where, se_u and p_u are the structure embedding and the position embedding, respectively, which are introduced in section 5.3.2 and section 5.3.3, $\mathrm{q}_v = W_q \cdot h_v$, $\mathrm{k}_u = W_k \cdot h_u, u \in N(v)$, and $\mathrm{v}_u = W_v \cdot h_u, u \in N(v)$, W_q, W_k,W_v and W_c are learnable weights. $W_{att1} \in \mathbb{R}^{d_{in} \times d_{out}}$ and $W_{att2} \in \mathbb{R}^{d_{out} \times 1}$ are learnable weights, d_{in} and d_{out} are the dimensions of the W_{att1}.

The weighted sum aggregation is

$$z_v = \sum_{u \in N(v)} att_{vu} \cdot (\mathrm{v}_u + \mathrm{se}_u + p_u), \tag{5.51}$$

Integration The integration is defined as follows:

$$h'_v = \sigma\left(W_l \cdot \mathrm{cat}(h_v, z_v)\right). \tag{5.52}$$

where W_l is learnable weight.

The settings of networks is summarized in Table 5.10. The graph pooling layer is introduced section 5.4.3.

5.8.6 Experiments and analysis

Here, we do experiments and analyses the efficiency of each part.

The classification accuracy (OA) are shown in Table 5.11 and Fig.5.19. According to Fig.5.19, we observed that the result of SUBGC was 90.8% , and the result of DotGC was 90.7% . The subtraction attention function is a little better than the dot-production attention function. However, when we compare MTGC with SUBGC, the multi-type attention function increased the result by 0.5% . When we compare MTGC with DotGC, the multi-type attention function increased the result by 0.6% . Therefore, we think that utilizing multi-type attention functions simultaneously can get better attention weights than only utilize one-type attention function, due to the influence of the type of attention function is eliminated. When we compare the result of MTPAGC with MTGC, the position embedding increased the result by 1.3% . Furthermore, when we compare the result of MTSPAGC with MTPAGC, the structure embedding increased the result by 0.7% . These results provide the evidence that each part of our MTSPAGC is effective.

Table 5.10: The settings of the simplified point cloud classification network using MTSPAGC. k is the value of k-NN, w is the number of selected nodes. We set t, the number of heads, for the multi-head graph transformer introduced before.

Learning rate	0.00135		
Drop out	0.3		
t	2		
Graph pooling layername	k	w	[input channels, output channels]
Graph pooling 0	15	512	[64,64]
Graph pooling 1	15	128	[128,128]
Graph convolution layername	k	[input channels, output channels]	
MTSPAGC 0	[11,15]	[3,64]	
MTSPAGC 1	[11,15]	[64,64]	
MTSPAGC 2	[11,15]	[64,64]	
MTSPAGC 3	[11,15]	[64,128]	
MTSPAGC 4	[11,15]	[128,256]	
MTSPAGC 5	[11,15]	[256,256]	

Table 5.11: Comparison results of the 3D shape classification on the ModelNet benchmark. OA indicates the average accuracy of all test instances.

	Method	ModelNet40
		OA
Different Transformer Layer	SUBGC	90.8%
	DotGC	90.7%
	MTGC	91.3%
	MTPAGC	92.6%
	MTSPAGC	93.3%

5.9 Conclusion

We present a new attention-based spatial graph convolution method, i.e., MTSPAGC, in this paper. It calculates feature angles, feature distances, and relational embedding in the feature space. It also utilize multi-type attention functions in one step of GC. We present a graph classification GNN and a node classification GNN based on the MTSPAGC. Our method performs better than other methods, according to experiments. Therefore, we demonstrated that local structural information in feature space, as well as multi-type attention function, can assist improve the expressive of AGCs.

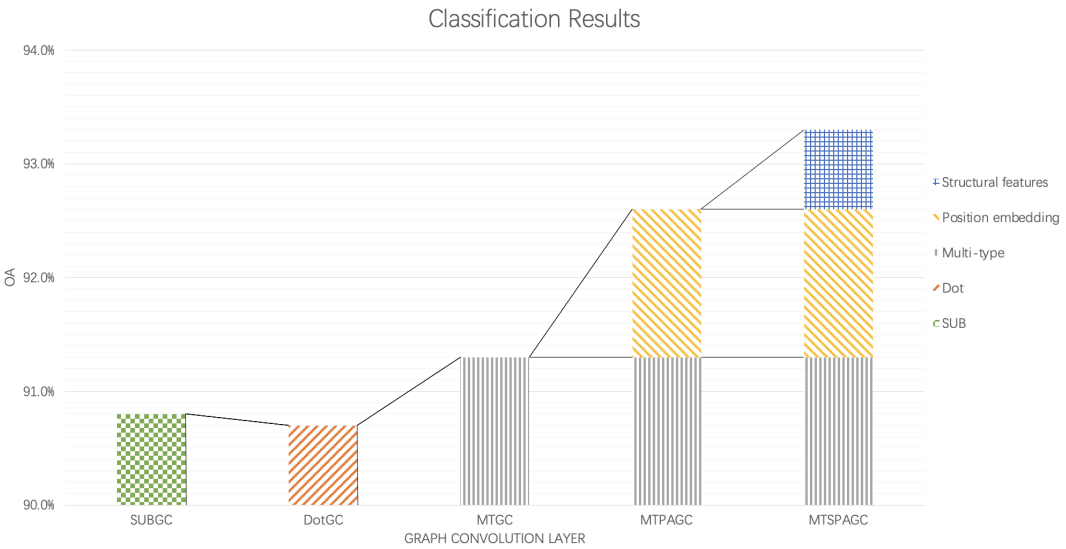

Fig. 5.19. The effect of different parts of our MTSPAGC. SUBGC is the subtraction attention-based graph convolution. It uses subtraction attention function to calculate attention weights and only uses node-wise features. DotGC is the dot-production attention-based graph convolution. It uses dot-production attention function to calculate attention weights and only uses node-wise features. MTGC is the multi-type attention-based graph convolution. It simultaneously uses subtraction and dot-production attention functions to calculate attention weights and only uses node-wise features. MTPAGC is multi-type position-aware attention-based graph convolution. It simultaneously uses subtraction and dot-production attention functions to calculate attention weights, and uses node-wise features and position embedding. MTSPAGC is our multi-type structure and position-aware attention-based graph convolution.

Chapter 6

General Conclusions

In this dissertation, we studied graph convolutions using local structures in feature space for graph neural networks. As we introduced in Chapter 1, the existing graph convolutions occur over-smoothing. Over-smoothing makes node-wise features of nodes where the edge exists be same. This may influence the performance of GNNs. We can reduce the occurrence of over-smoothing by improving the expressiveness of GCs. The expressiveness of GCs is that: If we can classify nodes by their node-wise features. Existing GCs have three expressive limitations:

1) Existing methods might lose the structural information of the surrounding neighboring nodes, particularly in the feature space.

2) While the multi-hop neighborhood may contain some useful information, this cannot be utilized for the single step graph convolution.

3) For attention-based spatial methods, they primarily use one type of attention function, i.e., dot-production, subtraction, and concatenation. However, different types of attention functions get different attention weights. These may influence the over-all performance of the method. Existing attention-based spatial methods do not consider this influence.

In Chapter 3, we focus on the limitation 1). We studied the problem how to describe the structure of neighboring nodes of a target node in feature space. Here, we proposed three structural features, i.e., feature angle, feature distance, and relational embedding. We introduced these structural features into GraphSAGE to propose a new spatial graph convolution, named structure-aware graph convolution (SAGConv). Based on the SAGConv, we proposed a graph classification network and a node classification network. During the experiments, our SAGconv has a better preference than other graph convolution methods. We also confirmed the effect of each structural feature. As a result, we think introducing structure information of neighboring nodes of a target node into a graph convolution can improve the expressive of graph convolution.

In Chapter 4, to further improve the expressive of GCs. based on the structural features and GraphSAGE [29], we proposed a new graph convolution, i.e., Structure-Aware Multi-Hop Graph Convolution (SAMGC). In SAMGC, we first utilize the structural features we proposed in Chapter 3. Then, our SAMGC gathers data from multi-hop neighboring nodes within one step of the graph convolution. Based on SAMGC, we proposed a graph classification network and a node classification network. Through experiments, our method performed better than other methods. We also conducted experiments to confirm how graph convolution expressive may be enhanced by local structural information in feature space as well as relevant information from indirected neighboring nodes.

In Chapter 5, we focus on limitation 3). We proposed a new attention-based graph convolution, i.e., Multi-type Structure and position-aware attention-based Graph Convolution (MTSPAGC). In MTSPAGC, we introduced the structural features, which proposed in Chapter 4, to our MTSPAGC. Then, we utilize multi-type attention functions in a single step of attention-based graph convolution. Based on MTSPAGC, we also proposed a graph classification network and a node classification network. Through experiments, our method is more powerful of expressive than existing methods. We also confirmed the effect of each part of our MTSPAGC.

We showed that utilize the structural information, the information of multi-hop neighborhood, and multi-type attention functions during a one-step of graph convolution, can improve the expressiveness of GCs in this dissertation. We are very glad to have contributed to this research field. How to better utilize the local structure information in the feature space, the information of multi-hop neighborhood, and multi-type attention functions during the graph convolution process are the main areas of interest in our subsequent research.

Acknowledgements

This dissertation was completed based on many supports from many nicest people. I would like to give my gratitude to these people.

First of all, I would like to express my sincere thanks to my supervisor, Professor Yuichi Tanaka. I met him in 2020. Since then he taught me many things regarding research, which include giving a presentation, writing an academic paper, and so on. These are definitely basis for my research career.

I am grateful to the laboratory members. They gave me many great advices and great supports.

At last, I would like to express my gratitude to my parents. They usually allowed me to make my own decision, and greatly supported me for my decision. Therefore, I can obtain a Ph.D degree. I could accomplish nothing without their support.

Reference

[1] T. Spangler, "Mrbeast ranks as highest-paid youtube star ever, earning an estimated 54 million in 2021," https://variety.com/2022/digital/news/mrbeast-highest-earning-youtube-star-54-million-1235154580/, jan 14, 2022.

[2] M. Vetterli, J. Kovacevic, and V. K. Goyal, *Foundations of Signal Processing.* Citeseer, 2014.

[3] S. Mallat, *A wavelet tour of signal processing.* Elsevier, 1999.

[4] E. C. Ifeachor and B. W. Jervis, *Digital signal processing: a practical approach.* Pearson Education, 2002.

[5] J. G. Proakis, *Digital signal processing: principles algorithms and applications.* Pearson Education India, 2001.

[6] K. R. Rao and P. Yip, *Discrete cosine transform: algorithms, advantages, applications.* Academic press, 2014.

[7] A. V. Oppenheim, *Discrete-time signal processing.* Pearson Education India, 1999.

[8] P. P. Vaidyanathan, *Multirate systems and filter banks.* Pearson Education India, 2006.

[9] G. Strang and T. Nguyen, *Wavelets and filter banks.* SIAM, 1996.

[10] R. Szeliski, *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[11] K. R. Castleman, *Digital image processing.* Prentice Hall Press, 1996.

[12] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still image data compression standard.* Springer Science & Business Media, 1992.

[13] D. S. Taubman and M. W. Marcellin, *JPEG2000: Image compression fundamentals.* Kluwer, 2002, vol. 11, no. 2.

[14] L. Deng, D. Yu *et al.*, "Deep learning: methods and applications," *Foundations and trends® in signal processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[15] J. Bruna, W. Zaremba, A. Szlam, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[16] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 779–788, 2016.

[18] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[19] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[20] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

[21] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," *Thirty-second AAAI conference on artificial intelligence*, 2018.

[22] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," *arXiv preprint arXiv:1709.04875*, 2017.

[23] W. Jin, K. Yang, R. Barzilay, and T. Jaakkola, "Learning multimodal graph-to-graph translation for molecular optimization," *arXiv preprint arXiv:1812.01070*, 2018.

[24] K. Han, Y. Wang, J. Guo, Y. Tang, and E. Wu, "Vision gnn: An image is worth graph of nodes," *arXiv preprint arXiv:2206.00272*, 2022.

[25] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[26] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[27] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[28] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[29] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, vol. 30, pp. 1024–1034, 2017.

[30] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[31] F. R. Chung, *Spectral graph theory*. American Mathematical Soc., 1997, no. 92.

[32] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in Neural Information Processing Systems*, vol. 29, pp. 3844–3852, 2016.

[33] 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.

[34] Y. Li and Y. Tanaka, "Structural features in feature space for structure-aware graph convolution," *2021 IEEE International Conference on Image Processing*, pp. 3158–3162, 2021.

[35] Y. Li and Y. Tanaka, "Structure-aware multi-hop graph convolution for graph neural networks," *IEEE Access*, vol. 10, pp. 16 624–16 633, 2022.

[36] D. Kim and A. Oh, "How to find your friendly neighborhood: Graph attention design with self-supervision," *arXiv preprint arXiv:2204.04879*, 2022.

[37] J. Yang, Z. Liu, S. Xiao, C. Li, D. Lian, S. Agrawal, A. Singh, G. Sun, and X. Xie, "Graphformers: Gnn-nested transformers for representation learning on textual graph," *Advances in neural information processing systems*, vol. 34, pp. 28 798–28 810, 2021.

[38] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *Advances in neural information processing systems*, vol. 34, pp. 28 877–28 888, 2021.

[39] D. Borrmann and J. Elseberg, "3d point cloud library," https://dx.doi.org/10.21227/H2S66T, 2018.

[40] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[41] L. Stanković, M. Daković, and E. Sejdić, "Introduction to graph signal processing," *Vertex-Frequency Analysis of Graph Signals*, pp. 3–108, 2019.

[42] A. Ortega, *Introduction to graph signal processing.* Cambridge University Press, 2022.

[43] S. Mcleod, "An easy guide to neuron anatomy with diagrams," https://https://www.simplypsychology.org/neuron.html, aug 9, 2022.

[44] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[45] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[46] S. Mallat, *A wavelet tour of signal processing.* Elsevier, 1999.

[47] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[48] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[49] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[50] J. Gehring, M. Auli, D. Grangier, and Y. Dauphin, "A convolutional encoder model for neural machine translation," *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 123–135, 2017.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[52] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 551–561, 2016.

[53] J. Q. Yang, D. C. Zhan, and X. C. Li, "Bottom-up and top-down graph pooling," *Advances in Knowledge Discovery and Data Mining*, pp. 568–579, 2020.

[54] Y. Tanaka, Y. C. Eldar, A. Ortega, and G. Cheung, "Sampling signals on graphs: From theory to applications," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 14–30, 2020.

[55] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *arXiv preprint arXiv:1511.06391*, 2015.

[56] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[57] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[58] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 3693–3702, 2017.

[59] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *Advances in Neural Information Processing Systems*, pp. 4800–4810, 2018.

[60] H. Gao and S. Ji, "Graph u-nets," *International Conference on Machine Learning*, pp. 2083–2092, 2019.

[61] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 723–731, 2019.

[62] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," *International Conference on Machine Learning*, pp. 3734–3743, 2019.

[63] G. Cheung, E. Magli, Y. Tanaka, and M. K. Ng, "Graph spectral image processing," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 907–930, 2018.

[64] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[65] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in Neural Information Processing Systems*, pp. 5099–5108, 2017.

[66] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1912–1920, 2015.

[67] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[68] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 652–660, 2017.

[69] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1492–1500, 2017.

[70] 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*, vol. 38, no. 5, pp. 1–12, 2019.

[71] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," *European Conference on Computer Vision*, pp. 483–499, 2016.

[72] J. Liu, B. Ni, C. Li, J. Yang, and Q. Tian, "Dynamic points agglomeration for hierarchical point sets learning," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 7546–7555, 2019.

[73] G. Te, W. Hu, A. Zheng, and Z. Guo, "Rgcnn: Regularized graph cnn for point cloud segmentation," *Proceedings of the 26th ACM international conference on Multimedia*, pp. 746–754, 2018.

[74] C. Wang, B. Samari, and K. Siddiqi, "Local spectral graph convolution for point set feature learning," *Proceedings of the European Conference on Computer Vision*, pp. 52–66, 2018.

[75] Y. Zhang and M. Rabbat, "A graph-cnn for 3d point cloud classification," *2018 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6279–6283, 2018.

[76] G. Pan, J. Wang, R. Ying, and P. Liu, "3dti-net: Learn inner transform invariant 3d geometry features using dynamic gcn," *arXiv preprint arXiv:1812.06254*, 2018.

[77] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining point cloud local structures by kernel correlation and graph pooling," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 4548–4557, 2018.

[78] 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.

[79] K. Hassani and M. Haley, "Unsupervised multi-task feature learning on point clouds," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 8160–8171, 2019.

[80] C. Chen, G. Li, R. Xu, T. Chen, M. Wang, and L. Lin, "Clusternet: Deep hierarchical cluster network with rigorously rotation-invariant representation for point cloud analysis," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 4994–5002, 2019.

[81] Q. Xu, X. Sun, C.-Y. Wu, P. Wang, and U. Neumann, "Grid-gcn for fast and scalable point cloud learning," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5661–5670, 2020.

[82] Y. Duan, Y. Zheng, J. Lu, J. Zhou, and Q. Tian, "Structural relational reasoning of point clouds," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 949–958, 2019.

[83] X. Yan, C. Zheng, Z. Li, S. Wang, and S. Cui, "Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 5589–5598, 2020.

[84] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 9621–9630, 2019.

[85] A. Komarichev, Z. Zhong, and J. Hua, "A-cnn: Annularly convolutional neural networks on point clouds," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7421–7430, 2019.

[86] Y. Rao, J. Lu, and J. Zhou, "Spherical fractal convolutional neural networks for point cloud recognition," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 452–460, 2019.

[87] J. Mao, X. Wang, and H. Li, "Interpolated convolutional networks for 3d point cloud understanding," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1578–1587, 2019.

[88] A. Boulch, "Convpoint: Continuous convolutions for point cloud processing," *Computers & Graphics*, vol. 88, pp. 24–34, 2020.

[89] K. Canese and S. Weis, "Pubmed: the bibliographic database," *The NCBI Handbook*, vol. 2, p. 1, 2013.

[90] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[91] Y. Liu, Y. Zhang, Y. Wang, F. Hou, J. Yuan, J. Tian, Y. Zhang, Z. Shi, J. Fan, and Z. He, "A survey of visual transformers," *arXiv preprint arXiv:2111.06091*, 2021.

[92] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 4700–4708, 2017.

[93] H. Zhao, J. Jia, and V. Koltun, "Exploring self-attention for image recognition," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 10 076–10 085, 2020.

[94] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 16 259–16 268, 2021.

[95] S. Xie, S. Liu, Z. Chen, and Z. Tu, "Attentional shapecontextnet for point cloud recognition," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 4606–4615, 2018.

[96] J. Yang, Q. Zhang, B. Ni, L. Li, J. Liu, M. Zhou, and Q. Tian, "Modeling point clouds with self-attention and gumbel subset sampling," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 3323–3332, 2019.

[97] J. Yang, Q. Zhang, B. Ni, L. Li, J. Liu, M. Zhou, and Q. Tian, "An end-to-end transformer model for 3d object detection," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2906–2917, 2021.

[98] N. Engel, V. Belagiannis, and K. Dietmayer, "Point transformer," *IEEE Access*, vol. 9, pp. 134 826–134 840, 2021.

[99] X. F. Han, Y. J. Kuang, and G. Q. Xiao, "Point cloud learning with transformer," *arXiv preprint arXiv:2104.13636*, 2021.

[100] X. F. Han, Y. F. Jin, H. X. Cheng, and G. Q. Xiao, "Dual transformer for point cloud analysis," *IEEE Transactions on Multimedia*, 2022.

[101] L. Wu, X. Liu, and Q. Liu, "Centroid transformers: Learning to abstract with attention," *arXiv preprint arXiv:2102.08606*, 2021.

[102] Y. Gao, X. Liu, J. Li, Z. Fang, X. Jiang, and K. M. S. Huq, "Lft-net: Local feature transformer network for point clouds analysis," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[103] M. H. Guo, J. X. Cai, Z. N. Liu, T. J. Mu, R. R. Martin, and S. M. Hu, "Pct: Point cloud transformer," *Computational Visual Media*, vol. 7, no. 2, pp. 187–199, 2021.

[104] D. Lu, Q. Xie, L. Xu, and J. Li, "3dctn: 3d convolution-transformer network for point cloud classification," *arXiv preprint arXiv:2203.00828*, 2022.

[105] J. Yu, C. Zhang, H. Wang, D. Zhang, Y. Song, T. Xiang, D. Liu, and W. Cai, "3d medical point transformer: Introducing convolution to attention networks for medical point cloud analysis," *arXiv preprint arXiv:2112.04863*, 2021.

[106] X. F. Han, Z. Y. He, J. Chen, and G. Q. Xiao, "3crossnet: Cross-level cross-scale cross-attention network for point cloud representation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3718–3725, 2022.

[107] Z. Cheng, H. Wan, X. Shen, and Z. Wu, "Patchformer: A versatile 3d transformer based on patch attention," *arXiv preprint arXiv:2111.00207*, 2021.