DHGCN: Dynamic Hop Graph Convolution Network for Self-supervised Point Cloud Learning

Jincen Jiang[1*], **Lizhi Zhao**[1*], **Xuequan Lu**[2†], **Wei Hu**[3], **Imran Razzak**[4], **Meili Wang**[1†],

[1]Northwest A&F University, [2]La Trobe University, [3]Peking University, [4]University of New South Wales
{jinec, zhaolizhi, wml}@nwsuaf.edu.cn, b.lu@latrobe.edu.au, forhuwei@pku.edu.cn, imran.razzak@unsw.edu.au

Abstract

Recent works attempt to extend Graph Convolution Networks (GCNs) to point clouds for classification and segmentation tasks. These works tend to sample and group points to create smaller point sets locally and mainly focus on extracting local features through GCNs, while ignoring the relationship between point sets. In this paper, we propose the Dynamic Hop Graph Convolution Network (DHGCN) for explicitly learning the contextual relationships between the voxelized point parts, which are treated as graph nodes. Motivated by the intuition that the contextual information between point parts lies in the pairwise adjacent relationship, which can be depicted by the hop distance of the graph quantitatively, we devise a novel self-supervised part-level hop distance reconstruction task and design a novel loss function accordingly to facilitate training. In addition, we propose the Hop Graph Attention (HGA), which takes the learned hop distance as input for producing attention weights to allow edge features to contribute distinctively in aggregation. Eventually, the proposed DHGCN is a plug-and-play module that is compatible with point-based backbone networks. Comprehensive experiments on different backbones and tasks demonstrate that our self-supervised method achieves state-of-the-art performance. *Our source code is available at*: https://github.com/Jinec98/DHGCN.

1 Introduction

A point cloud is an unordered collection of scattered points representing geometric information in the 3D space. Point cloud processing and understanding are crucial in many areas, such as autonomous driving, virtual reality, etc. Unlike regular 2D pixels on the images, point clouds typically have irregular point distributions, making it difficult to directly apply traditional Convolution Neural Networks (CNNs) to point clouds for extracting features (**???**).

In addition, the complex geometric structures of the point cloud can be well-structured with graphs by encoding the representations of pairwise relationships of points (**??**). Therefore, researchers have made efforts to generalize Graph Convolution Networks (GCNs) to point clouds for classification and segmentation, achieving encouraging results (**????**). DGCNN (**?**) constructs a k-Nearest Neighbor (kNN) graph for each center point, and captures graph structure by propagating and aggregating the offset relationships between a center point and its neighbors. **?** assigns attention weights to different neighboring points and feature channels. These works tend to sample and group points to create point sets locally, and construct a graph for each set (Figure 1 (d)-(e)). They usually focus on extracting graph representations for local point sets through GCNs, while ignoring the explicit contextual relationships between them.

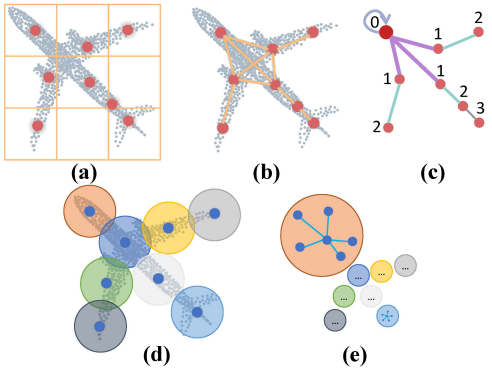

Figure 1: **First row:** Constructing the ground truth graph for *our self-supervised hop distance reconstruction task*. (a): Voxelizing the point cloud into parts, taking each part as a graph node. (b): The topology of the ground truth graph. Two nodes are adjacent if their scaled bounding boxes are intersected. (c): The shortest path between a node (enlarged red point) and other nodes. The number on each node denotes the hop distance which motivates *our self-supervised task*. **Second row:** Sampling and grouping based strategy (**??**). (d): Sampling center points and grouping local point sets. (e): Constructing a local graph for each point set. We explore the contextual relationships between parts, while previous strategies focus on extracting local features of point sets.

In this paper, we attempt to embed the distance in geometric space between point sets explicitly to learn their contextual relationships (e.g., topology, adjacency, etc). Our intuition is that the geometric distance of point sets lies in the

*These authors contributed equally.
†Corresponding authors.

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

pairwise adjacent relationship between them. In addition, with considering point sets as nodes and representing the point cloud as a graph, the hop distance concept in graph theory can depict the degree of adjacency quantitatively.

Motivated by this intuition, we propose a novel *self-supervised part-level hop distance reconstruction task*. We design the Dynamic Hop Graph Convolution Network (DHGCN) for extracting and embedding the low-level geometric distance to learn the high-level contextual relationships between voxelized point parts. In the pre-processing stage, we first split the entire point cloud into voxel parts, and construct a *ground truth graph* with each part serving as a graph node to compute the distance matrix, which can imply the degree of adjacency between two nodes quantitatively (Figure 1 (a)-(c)). Next, given an input point cloud, we construct a complete graph with randomly initialized distance matrix (i.e., no contextual information) and attempt to predict the hop distance matrix to learn the part-level contextual relationship. We devise a hop distance loss to supervise the predicted distance matrix that is dynamically updated in each layer. Further, we design Hop Graph Attention (HGA) that takes the learned distance matrix as input for assigning more attention to edge features between neighboring parts (i.e., parts with short distances) and less attention to distant parts, allowing edge features to contribute distinctively in aggregation. Finally, we make our DHGCN compatible with point-based backbone networks through pooling and repeating, making it a plug-and-play module.

The main contributions of this paper are as follows.

- We propose a novel self-supervised hop distance reconstruction task and a hop distance loss for learning the contextual relationships between point parts, by considering the hop distance as the proxy to depict the degree of adjacency quantitatively.
- We propose Hop Graph Attention, a module that takes the dynamically learned hop distance as input to produce attention weights, allowing edge features to contribute distinctively in aggregation.
- We make our DHGCN a plug-and-play module that can be easily embedded in point-based backbone networks. Extensive experiments show that our self-supervised DHGCN achieves state-of-the-art performance on different downstream tasks.

2 Related Work

2.1 Self-supervised Point Cloud Learning

Point-based methods are pioneered by PointNet (**?**), which consumes raw point cloud as input using shared multi-layer perceptions (MLPs). PointNet++ (**?**) further devises a hierarchical architecture that recursively samples point sets to capture multi-scale local geometric information. As the cornerstone of point-based methods, PointNet++ has inspired numerous modern works.

Self-supervised learning (SSL) methods for point clouds aim to learn point cloud intrinsic representations through well-designed pretext tasks without labeled data. Recent works can be roughly summarized as contrastive and reconstructive methods (**?**). The contrastive methods contrast the latent representations of different point cloud transformation views (e.g., rotation, jitter, scale, etc.) and design pretext tasks based on inter-data information such as similarity (**???**). CrossPoint (**?**) enforces the correspondence between a point cloud and its rendered 2D image while preserving the model's invariance to spatial transformations. The reconstructive methods typically aim to reconstruct intra-data information from low-quality (e.g., mask, noise, etc.) input, which exploits the point cloud intrinsic structure as self-supervised signals (**???**). OcCo (**?**) generates occluded point clouds from randomly sampled camera views and trains an encoder-decoder model to complete the original point clouds.

2.2 Graph-based Learning Methods

Graphs are the universal representations of heterogeneous pairwise relationships of non-Euclidean data, such as point clouds (**?**). Previous works extend convolution from 2D CNNs to graphs by processing the graph spectral representation (**??**). DGCNN (**?**) constructs kNN graphs for investigating correlated relationships among neighboring points with the EdgeConv operation, which encodes the graph local geometric information by propagating and aggregating edge features. 3D-GCN (**?**) proposes the deformable GCN kernel with learnable shapes and weights. AdaptConv (**?**) generates adaptive kernels for convolution on mutually correlated point pairs according to their edge features. More recently, **?** transfers residual/dense connections and dilated convolutions to GCNs to train very deep GCNs, avoiding vanishing gradients. These works usually partition a point cloud into point sets by the kNN or radius ball query method, and design sophisticated local feature extractors for learning point sets features.

2.3 Attention Mechanism

Attention mechanism exhibits the ability to extract relationships between representations by focusing on the most relevant parts of inputs to make decisions (**?**). **?** proposes Transformer, a model architecture relying solely on a self-attention (SA) mechanism, pioneering follow-up SA based works. Graph attention network (GAT) (**?**) computes the hidden representations of each graph node by attending to its neighbors' features with assigned attention weights. **?** proposes Graph attention convolution, implicitly assigning different attention importance to neighboring nodes and feature channels. **?** proposes PCT, which applies the offset-attention Transformer to learn the local context of the point cloud.

Unlike previous works that focus on designing sophisticated encoders for learning local features of grouped point sets, we propose the HGA to learn the contextual relationship between point sets, explicitly providing geometric information. This is driven by the fact that the inherent graph structure depicts the adjacent relationship between parts.

3 Method

The overall architecture of the DHGCN is shown in Figure 2. DHGCN aims to learn the high-level contextual information from the pairwise neighboring relationship between

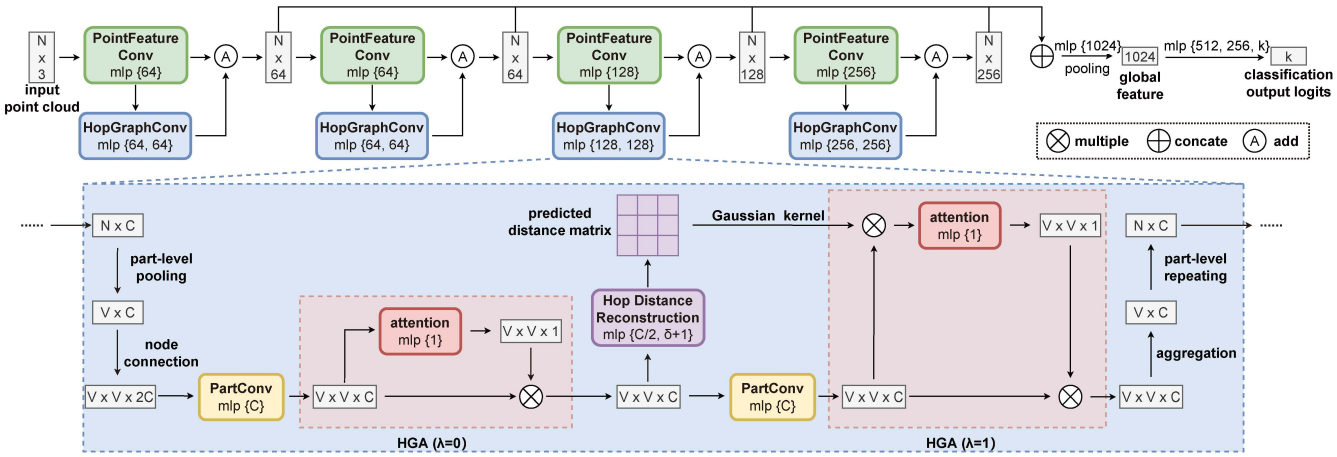

Figure 2: **DHGCN architecture**: We feed the input point cloud to PointFeatureConv for extracting point-wise representations, which are then taken as input by Hop Graph Convolution (HopGraphConv), to extract more accurate local geometric representations. **Hop Graph Convolution**: The HopGraphConv layer takes the point features as input, and achieves part features through part-level pooling. We construct a complete graph by taking parts as nodes and connecting each pair of them, and use PartConv and HGA to extract graph edge features. We propose the self-supervised hop distance reconstruction task to predict the distance matrix of the complete graph from edge features. λ controls whether the HGA embeds hop distance. Finally, edge features are aggregated and repeated at the part-level, providing additional representations for the point-based backbone network.

point parts. We first voxelize the entire point cloud into parts and construct a graph by considering each part as a node. We define a distance matrix on the graph as the self-supervised signal, implying the degree of adjacency between nodes. Our DHGCN extracts parts features while also predicting the hop distance matrix, which is taken as input by the Hop Graph Attention (HGA) to embed the learned geometric information into point features.

3.1 Volumetric Partition

This section discusses the details of point cloud volumetric partition and part-level graph construction. Given an input point cloud $\mathcal{X} = \{x_i \mid i = 1, 2, \ldots, N\} \in \mathbb{R}^{N\times 3}$ with N points, we voxelize it by mapping each point to a voxel part. As a result, the volumetric partition splits $\mathcal{X}$ into $V = s^3$ parts: $\mathcal{P} = \{p_i \mid i = 1, 2, \ldots, V\}$, where s is the split number, V is the number of parts, and p_i is a list containing the indices of points inside the i-th part. For each part, we compute its up-scaled axis-aligned bounding box with the scale factor set to 1.2 for calculating the adjacency between parts in the geometric space. For parts with no points inside, we simply set its bounding box volume to 0.

We can now construct a ground truth graph $G = (\mathcal{V}, \mathcal{E})$ with each part serving as a graph node. We define two nodes are connected by an edge if their bounding boxes are intersected. We also introduce self-loops, connecting each node with itself. The distance between two nodes is defined as the number of edges (i.e., hops) in their shortest path. Therefore, G's distance matrix $D \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ implies the pairwise degree of adjacency between nodes quantitatively. The maximum distance δ is truncated to $s + 1$ hops and the self-loop distance is defined as 0 hop.

3.2 Part Feature Extraction

Point-based networks such as PointNet (**?**), DGCNN (**?**) and AdaptConv (**?**) usually extract point-wise representations in each layer by using a point feature convolution (PointFeatureConv) function g_p with learnable parameters to map the input point cloud representations to a new set of C-dimensional point features $\mathcal{H} = \{h_i \mid i = 1, 2, \ldots, N\} \in \mathbb{R}^{N\times C}$. Then the point cloud's global features $f_g \in \mathbb{R}^C$ are usually derived by applying the permutation-invariant max pooling to aggregate point-wise features globally:

$$f_g = \max_{j\in\{1,2,\ldots,N\}} h_j. \tag{1}$$

In our DHGCN framework, we expect to achieve part-level features from the input point-wise features $\mathcal{H}$. Similarly, we pool the point-wise features in *part-level* to obtain part features $\mathcal{F} = \{f_i \mid i = 1, 2, \ldots, V\} \in \mathbb{R}^{V\times C}$, where

$$f_i = \max_{j\in p_i} h_j \tag{2}$$

and p_i contains the indices of points in the i-th part as described in Section 3.1.

3.3 Hop Graph Convolution Module

Part Convolution Given the input parts $\mathcal{P}$ with its corresponding part features $\mathcal{F}$, we initialize a complete graph $\tilde{G} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ whose topology implies no specific contextual information, with parts serving as graph nodes for applying part-level graph convolution (PartConv). Inspired by DGCNN (**?**), we define the edge feature between the i-th and j-th node as:

$$e_{ij} = [f_i, f_j - f_i] \in \mathbb{R}^{2C}, \tag{3}$$

where $[\cdot,\cdot]$ is the concatenation operation. e_{ij} explicitly combines shape information with neighborhood information encoded by the feature differences (**?**). Our PartConv projects e_{ij} to a new set of edge features e'_{ij}, attempting to extract more accurate local geometric representations (**?**) as follows:

$$e'_{ij} = g_m\left(e_{ij}\right), \tag{4}$$

where $j \in \mathcal{N}(i) = \{j : (i,j) \in \tilde{\mathcal{E}}\}$, $\mathcal{N}(i)$ is node indices connected with the i-th node. $g_m(\cdot) : \mathbb{R}^{C_{in}} \rightarrow \mathbb{R}^C$ is a MLP with learnable parameters.

Hop Distance Reconstruction Since the low-level geometric distance can be represented by the degree of adjacent relationship between parts, we extract it by reconstructing the hop distance matrix of $\tilde{G}$ with the proposed self-supervised task. We denote the predicted hop distance of the i-th and j-th nodes of $\tilde{G}$ as $\tilde{D}_{ij}$ and consider predicting $\tilde{D}_{ij}$ as a classification problem of $\delta + 1$ categories (including self-loops), where δ is the maximum distance as defined in Section 3.1. In this sense, we predict $\tilde{D}_{ij}$ by applying the MLP g_h to edge features e'_{ij} as:

$$\tilde{D}_{ij} = g_h(e'_{ij}). \tag{5}$$

As shown in Figure 3, the predicted distance matrix $\tilde{D}$ is updated in each layer supervised by the ground truth distance matrix D. Concretely, we propose the hop distance loss $\mathcal{L}_h$ to measure the discrepancy between $\tilde{D}$ and D in each layer:

$$\mathcal{L}_h = \sum_i^V \sum_j^V \mathrm{CE}(\tilde{D}_{ij}, D_{ij}), \tag{6}$$

where CE is a cross-entropy function.

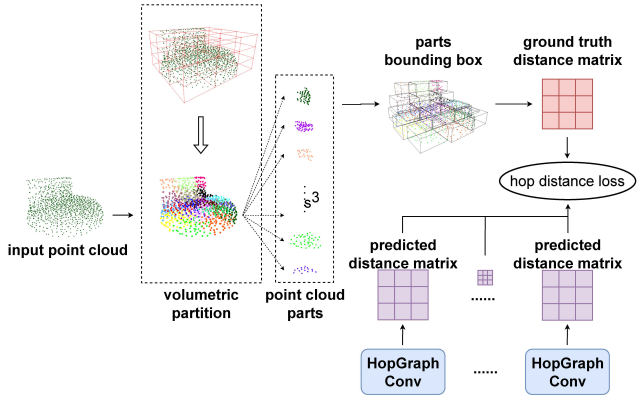

Figure 3: Given the input point cloud, we first voxelize it into parts. For each part, we compute its scaled axis-aligned bounding box to calculate the adjacent relation between parts. We construct a ground truth graph along with its distance matrix for supervision in each layer.

Hop Graph Attention The Hop Graph Attention aims to embed the learned geometric structure information into high-level point cloud contextual features by assigning more attention weights to edge features between neighboring parts in the geometric space (i.e., parts with low hops). Therefore, we compute the hop attention coefficient of e'_{ij} as follows:

$$t_{ij} = g_a\left(\lambda \cdot \mathbb{G}(\tilde{D}_{ij})e'_{ij} + (1-\lambda)e'_{ij}\right), \tag{7}$$

where $j \in \mathcal{N}(i)$, and $g_a : \mathbb{R}^C \rightarrow \mathbb{R}$ is a shared attention MLP. $\mathbb{G}(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}x^2\right)$ represents Gaussian kernel. $\lambda \in \{0,1\}$ is a switch factor that controls whether to embed the hop distance. When $\lambda = 0$, the HGA equals to general SA to extract preliminary features.

The hop attention coefficient t_{ij} indicates the learned importance of part p_j to p_i, which is inferred from the predicted hop distance $\tilde{D}_{ij}$. Since the hop distance depicts the adjacent relationship between parts, explicitly multiplying $\tilde{D}_{ij}$ by the edge features e'_{ij} embeds the low-level geometric distance into the high-level contextual features, producing more expressive attention weights.

We normalize coefficients using softmax function to make it comparable across all connected neighbors as follows:

$$\alpha_{ij} = \mathrm{softmax}_j\left(t_{ij}\right) = \frac{\exp\left(t_{ij}\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(t_{ik}\right)} \tag{8}$$

We can now calculate the learned part features $\tilde{f}_i$ by aggregating edge features between the i-th node and all other connected nodes distinctively:

$$\tilde{f}_i = \max_{j\in\mathcal{N}(i)} \alpha_{ij} \cdot e'_{ij}. \tag{9}$$

Revising Point-wise Features For each part-level feature $\tilde{f}_i$, we repeat it $|p_i|$ times to align it with corresponding points, constructing the revised point-wise features $\tilde{\mathcal{H}} = \{\tilde{h}_i \mid i = 1, 2, ..., N\}$ by fusing with the original point-wise features $\mathcal{H}$ through element-wise addition for providing more expressive representations for our point-based backbone network:

$$\tilde{h}_j = \tilde{f}_i + h_j, j \in p_i. \tag{10}$$

Figure 4 visualizes our attention maps and the comparison of feature distance between our method and the DGCNN backbone. The attention maps (row 1) depict that the predicted hop distance is consistent with the geometric relationship of each part, which enables our learned features $\tilde{\mathcal{H}}$ (row 2) to be more distinctive than DGCNN (row 3), and more related to both geometric distance and shape information.

Dynamic Hop Distance Previous point learning works (**???**) tend to construct a local graph for each selected center point through kNN in feature space, and update center points and corresponding graphs dynamically in each layer. Since their edge features contribute equally in aggregation, these methods can be viewed as learning local features from local point sets using a GCN with a local receptive field.

Different from these works, we construct a complete global graph $\tilde{G}$ rather than a local graph, allowing every

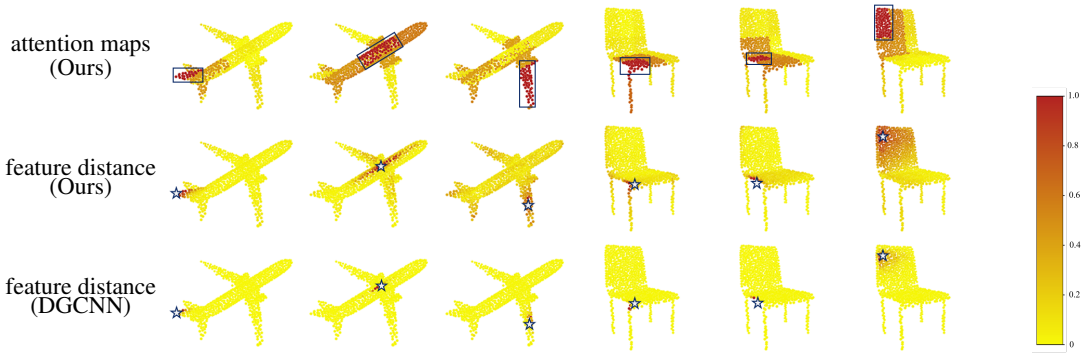

Figure 4: Attention maps (row 1) for different query parts and feature distance from the query point (indicated by star, row 2 (ours) and row 3 (DGCNN)) to all other points, with yellow to red indicating increasing attention weight or closer distance.

node to attend to every other node, making the receptive field cover the entire point cloud globally. With the aid of the distance reconstruction task, $\tilde{G}$'s hop distance matrix is dynamically updated in each layer, which is enabled by the Gaussian kernel for providing more attention to neighboring parts and less attention to distant parts, allowing edge features to contribute distinctively in aggregation.

We calculate the proposed hop distance loss $\mathcal{L}_h^{(l)}$ in each layer to provide strong supervision for updating the distance matrix. Finally, the self-supervised loss of our DHGCN for the hop distance prediction task is defined as:

$$\mathcal{L} = \sum_{l}^{L} \mathcal{L}_h^{(l)}, \tag{11}$$

where L denotes the number of HopGraphConv layers.

4 Experimental Results

4.1 Experimental Setting

We implement our method in PyTorch. The SGD optimizer is used for all experiments. We use one TITAN RTX GPU for training. The training batch size is set to 32. Following DGCNN, we set the dropout rate to 0.5 and the random seed to 1. The learning rate is set to 0.1 under the cosine learning scheduler with the 0.9 momentum and the 0.0005 weight decay. For classification, we set the split number $s = 3$ by default, i.e., the point cloud is split into $3^3 = 27$ parts, and the max distance $\delta = 4$, i.e., the hop distance will be classified into 5 categories (including hop = 0), and we set $s = 5$ for segmentation tasks for a more fine-grained splitting. We use 4 heads for multi-head attention in the Hop Graph Attention module. σ in Gaussian kernel is set to 1 by default.

4.2 Pretraining

Pretrained datasets. Our DHGCN uses ShapeNet (**?**) for pretraining. The dataset has $57,448$ models with 55 categories, and all models will be used for our self-supervised pretraining task. Following previous work, we use $2,048$ points as input. Note that some methods are pretrained on ModelNet40 (**?**). We also follow this setting, and use the train set ($9,840$ training models) of ModelNet40 for pretraining. As for ModelNet40, we use $1,024$ points as input which is the same with GraphTER.

Training details. The DHGCN is a part-level plug-and-play module that is compatible with point-based backbone networks, by implementing the PointFeatureConv with backbone modules. We use several different point-based backbone networks under the linear protocol of unsupervised representation learning to verify the effectiveness of our self-supervised reconstruction task. This training strategy is a two-stage paradigm. The first stage is to pretrain the network with only the proposed self-supervised task, and the second stage is to *freeze* the pretrained model and train a linear classifier only for downstream tasks, i.e., 3D object classification and shape part segmentation.

4.3 Downstream Tasks

3D object classification on ModelNet40. We use ModelNet40 for 3D point cloud classification. Data split protocols following PointNet. The dataset contains $9,840$ models for training and $2,468$ models for testing, involving a total of 40 categories. The sampling strategy in PointNet is adopted to sample each point cloud into $1,024$ points. We only use normalized coordinates as input without considering normals.

Table 1 shows the comparison results of our method and the SOTA unsupervised methods. We divide the results based on two pretrained datasets. As we can see, our DHGCN notably outperforms recent works, achieving SOTA performance on both pretrained datasets (93.2% for ShapeNet and 93.3% for ModelNet40), exceeding SSC by 0.8% with pretraining on ShapeNet and surpassing GraphTER by 1.3% with pretraining on ModelNet40.

Classification with limited data. Following previous works (**?**), we further train a linear classifier with only limited sampled training data of ModelNet40 to evaluate the pretrained model's generalizability on all test data. As shown in Table 2, our method with DGCNN as backbone can achieve 89.1% with 20% data used for training and 86.1% with 10% data. In the extreme case of using only 1% data, our DHGCN achieves 62.7% accuracy, exceeding that of MAE3D by 1% and FoldingNet by 6.3%. Our DHGCN

Methods	Pretrained dataset	# Points	Acc.	Methods	Pretrained dataset	# Points	Acc.
LatentGAN (**?**)	SN	2k	85.7	FoldingNet (**?**)	MN	2k	84.4
FoldingNet (**?**)	SN	2k	88.4	LatentGAN (**?**)	MN	2k	87.3
PointCapsNet (**?**)	SN	2k	88.9	PointCapsNet (**?**)	MN	1k	87.5
VIPGAN (**?**)	SN	2k	90.2	Multi-task (**?**)	MN	2k	89.1
STRL (**?**)	SN	2k	90.9	MAP-VAE (**?**)	MN	2k	90.2
SSC (RSCNN) (**?**)	SN	2k	92.4	GraphTER (**?**)	MN	1k	92.0
CrossPoint (**?**)	SN	2k	91.2	GLR (RSCNN) (**?**)	MN	1k	92.2
DHGCN (DGCNN)	SN	2k	**93.2**	DHGCN (DGCNN)	MN	1k	93.0
DHGCN (AdaptConv)	SN	2k	**93.2**	**DHGCN (AdaptConv)**	MN	1k	**93.3**

Table 1: Classification results of *unsupervised* methods (including ours) on ModelNet40. 'SN/MN' denotes 'ShapeNet/ModelNet40' and '# Points' indicates the point number in pretraining.

Methods	Limited training data ratios				
	1%	2%	5%	10%	20%
FoldingNet (**?**)	56.4	66.9	75.6	81.2	83.6
MAE3D (**?**)	61.7	69.2	80.8	84.7	88.3
DHGCN	**62.7**	**72.2**	**81.3**	**86.1**	**89.1**

Table 2: Comparison results of 3D object classification with limited training data (different ratios) on ModelNet40. DGCNN is taken as the backbone.

achieves SOTA results in all 5 training data ratios, demonstrating that the features learned by our self-supervised task can be easily generalized to the point cloud classification task even with limited training data.

Classification on real-world dataset ScanObjectNN. We also conduct the classification experiment on the real-world scanning dataset ScanObjectNN (**?**), which poses great challenges for point cloud classification methods due to the involved cluttered background, noisy perturbations and occluded incomplete data. This dataset contains 15 categories, totally $2,902$ unique object instances. Here we follow the official data split strategy on three dataset variants: OBJ_ONLY, OBJ_BG and PB_T50_RS, and conduct pertaining on ShapeNet.

As shown in Table 3, DHGCN achieves the best results of 85.9% on OBJ_BG variant, exceeding all compared SOTA unsupervised methods by at least 4.2%. Our DHGCN even achieves comparable results with supervised methods, e.g., surpassing the DGCNN backbone by 3.1% on OBJ_BG. As for the PB_T50_RS variant, our method achieves an accuracy of 81.9%, slightly lower than the other two variants. We suspect that the perturbation noise in PB_T50_RS disturbs the adjacency relationship between parts (i.e., some points are incorrectly split into adjacent parts during voxelization), thus degrading the power of the learned hop distance in depicting point cloud intrinsic geometric structure.

Furthermore, despite being pretrained on the ShapeNet dataset (synthetic data), our downstream classification results on the real-world ScanObjectNN reveal that the learned geometric information is useful in mitigating the domain gap between synthetic and real-world data.

Methods	Sup.	OBJ_ONLY	OBJ_BG	PB_T50_RS
PointNet (**?**)	✓	79.2	73.3	68.2
PointNet++ (**?**)	✓	84.3	82.3	77.9
PointCNN (**?**)	✓	85.5	86.1	78.5
DGCNN (**?**)	✓	86.2	82.8	78.1
Point-BERT (**?**)	✓	88.1	87.4	83.1
Point-MAE (**?**)	✓	88.3	90.0	85.2
Jigsaw (**?**)	✗	-	59.5	-
OcCo (**?**)	✗	-	78.3	-
STRL (**?**)	✗	-	77.9	-
CrossPoint (**?**)	✗	-	81.7	-
DHGCN	✗	**85.0**	**85.9**	**81.9**

Table 3: Classification results of our method and state-of-the-art methods on ScanObjectNN. DGCNN is used as backbone. 'Sup.' denotes the method is supervised (✓) or unsupervised (✗). Results of Jigsaw, OcCo, and STRL are from CrossPoint, and "-" indicates no previous results.

Part segmentation on ShapeNet Part. We evaluate DHGCN for the shape part segmentation task on ShapeNet Part dataset (**?**), which contains $16,881$ models from 16 categories. Each model involves 2 to 6 parts, with a total number of 50 distinct part labels. Following PointNet, we sample or interpolate each model to $2,048$ points and only use point coordinates as input.

We use mean Intersection-over-Union (mIoU) as the evaluation metric, and two types of mIoU are reported in Table 4. Our self-supervised method achieves SOTA performance, which exceeds all recent unsupervised methods.

Part segmentation with limited data. We *freeze* the pretrained model and randomly sample 1% and 5% of the train set of ShapeNet Part to train several MLPs for evaluating the segmentation task with the unsupervised paradigm. Results shown in Table 5 demonstrate that our DHGCN using PAConv as backbone achieves SOTA performance, i.e., 76.9% with 1% training data and 81.9% with 5% training data for instance mIoU, which exceeds recent unsupervised methods. These results reveal that the features learned by our self-supervised hop distance reconstruction task are more expressive than other unsupervised methods in terms of part segmentation, under extremely limited training data.

Methods	Sup.	Class mIOU	Instance mIOU
PointNet (?)	✓	80.4	83.7
PointNet++ (?)	✓	81.9	85.1
DGCNN (?)	✓	82.3	85.2
KPConv (?)	✓	85.1	86.4
PAConv (?)	✓	84.2	86.0
Point-BERT (?)	✓	84.1	85.6
LatentGAN (?)	✗	57.0	-
MAP-VAE (?)	✗	68.0	-
GrpahTER (?)	✗	78.1	81.9
CTNet (?)	✗	75.5	79.2
DHGCN	✗	**82.9**	**84.9**

Table 4: Shape part segmentation results of our method and state-of-the-art techniques on ShapeNet Part dataset. PAConv is used as backbone. 'Sup.' denotes the method is supervised learning (✓) or unsupervised learning (✗).

Methods	Limited training data ratios	
	1%	5%
SO-Net (?)	64.0	69.0
PointCapsNet (?)	67.0	70.0
Multi-task (?)	68.2	77.7
PointContrast (?)	74.0	79.9
SSC (RSCNN) (?)	74.1	80.1
DHGCN	**76.9**	**81.9**

Table 5: Comparison results of shape part segmentation with limited training data (different ratios) on ShapeNet Part. PAConv is taken as the backbone.

Please refer to supplementary for more results.

4.4 Ablation Studies

Attention mechanism. We conduct an ablation study for several model settings to verify the HGA's effectiveness, which embeds the learned hop distance matrix into edge weights. We denote the SA option as our baseline, whose switch factor λ is set to 0 in both HGA layers. HGA will degrade to general SA in this case, and the hop distance loss is disabled. Note that we train this baseline in a *supervised* manner. We compare the effect of whether the hop distance loss is calculated in each layer or only the last layer. Accuracy results of point cloud classification and hop distance prediction are reported in Table 6.

With the aid of HGA, the classification accuracy significantly exceeds that of SA baseline by 0.5%. In addition, calculating the hop distance loss in each layer leads to both higher hop distance prediction accuracy (94.6% versus 93.3%) and point cloud classification accuracy (93.3% versus 93.1%). The strong supervision of our hop distance loss leads to a more accurate learned hop distance matrix, thus producing better performance.

Gaussian kernel. The predicted hop distance will be processed by the Gaussian kernel $\mathbb{G}$ to provide more attention to neighboring parts (i.e., smaller distances yield higher weights and vice versa). The parameter σ^2 of $\mathbb{G}$ controls the decay rate between distance and edge weight. A small

Attention	Sup.	Loss	Distance Acc.	Acc.
SA	✓	$\mathcal{L} = \mathcal{L}_c$	-	92.8
HGA	✗	$\mathcal{L} = \mathcal{L}_h^{(-1)}$	93.3	93.1
HGA	✗	$\mathcal{L} = \sum_l^L \mathcal{L}_h^{(l)}$	**94.6**	**93.3**

Table 6: Different attention mechanisms. Experiments are conducted on ModelNet40 with AdaptConv as the backbone. SA denotes self-attention, and HGA denotes Hop Graph Attention. Accuracy results of hop distance prediction and point cloud classification are reported.

Gaussian kernel					
σ^2	0.2	0.5	**1.0**	2.0	5.0
Acc.	92.6	93.2	**93.3**	93.0	92.9

Table 7: Ablation results on different σ^2 in the Gaussian kernel. Experiments are conducted on ModelNet40 for classification with AdaptConv as the backbone.

σ^2 causes the edge weights between remote parts to decay rapidly. For example, when $\sigma^2 = 0.2$, the weights of parts over 1-hop distance converge to 0. At this point, the receptive field degrades to 1-hop (i.e., local neighbors), resulting in a lower accuracy of 92.6%, as shown in Table 7. On the contrary, with a larger σ^2, the weights decay gently as the distance increases, enabling nodes to contribute more equally. However, this leads to a reduction in distinction due to distance, achieving only 93.0% ($\sigma^2 = 2.0$) and 92.9% ($\sigma^2 = 5.0$). The model achieves the highest accuracy of 93.3% when $\sigma^2 = 1.0$.

5 Conclusion

This paper proposes a novel self-supervised part-level hop distance reconstruction task and a novel hop distance loss to learn contextual relationships between point parts. The dynamically updated hop distances are embedded as attention weights by the proposed HGA for determining point parts' importance in feature aggregation. Our DHGCN can be easily incorporated into point-based backbones. We outperform SOTA unsupervised methods on both downstream classification and part segmentation tasks. Our model is less effective for data with large perturbations as noise leads to less accurate splitting of parts, which tends to produce misleading adjacent relationships. This will be explored in future.

Acknowledgments

This work is supported in part by the National Key Research and Development Program of China (Grant Number 2022ZD04014) and the Shaanxi Province Key Research and Development Program (Grant Number 2022QFY11-03).

illo quod corporis dolore, obcaecati iste facilis consequatur labore quisquam perspiciatis dolor, numquam exercitationem laudantium ea aliquam, reiciendis minima dignissimos asperiores?Veritatis incidunt repellendus dignissimos, odio accusamus harum unde vitae in quos, debitis provident quidem atque nihil, doloribus itaque nulla magni enim expedita velit?