Learning Disentangled Representations of Point Clouds via Alpha Complexes for 3D Shape Classification

ALTAF M. AGOWUN

Submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science, Honours Computing Science

at

Saint Mary's University
Halifax, Nova Scotia

Approved by: Dr. Jiju Poovvancheri
Supervisor

Dr. Somayeh Kafaie
Reader

Date: April 26, 2024

Abstract

Learning Disentangled Representations of Point Clouds via Alpha Complexes for 3D Shape Classification

By

Altaf M. Agowun

Three-dimensional computer vision tasks have gained much attention in recent times, both in academic and industrial research. One of the key tasks of 3D computer vision is object classification. Various approaches based on the representations (e.g., point clouds, voxels, multi-view images and graphs) of the objects have been put forward for object classification. Recently, few works have used graph neural network for point cloud classification and have achieved promising results. In this thesis, we explore the use of a dual-stream graph neural network combining the alpha complexes constructed on the feature and non-feature regions of the point cloud object. The disentangled representation of the point cloud into feature and non-feature regions is achieved through a gradient structure analysis procedure and a Corner and Edge detection technique. Our experiments on ModelNet40 benchmark dataset indicate that the proposed graph-based method achieves higher or comparable accuracy to other state-of-the-art methods.

Date: April 26, 2024

Acknowledgments

I would like to thank Dr. Jiju Poovvancheri for guiding me and giving me the opportunity to start my research career. I am very grateful for his constant support and encouragement that empowered me to grow and take on opportunities I would not have taken otherwise. I would also like to thank Ms. Prachi Kudeshia for all the valuable help in starting my research career and making this thesis possible. Above all I am forever grateful for such amazing parents without whom I would not have been able to achieve anything in life. Finally, I would like to thank everyone who directly or indirectly played a role in my success so far.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Three-dimensional, (3D) object classification is the task of predicting the class of a 3D object. In recent years, we have seen a lot of promising results in object classification within two-dimensional images [8–10], however the environment we interact with is in three-dimensional and 2D data lack important information such as depth which is especially important when looking at real-world applications of such research such as autonomous driving, robotics or Augmented Reality. With the development of affordable 3D sensors (LiDAR sensors, RGBD-sensors, etc.) together with the availability of public datasets (ShapeNet [1], ScanObjectNN [11], etc.) research within 3D learning tasks such as object classification, object segmentation, object detection, etc. have been gaining a lot of interest. In this thesis, we focuses on the task of point cloud classification using GNN, more specifically we explore the use of Geometric Disentangled representation and alpha complex graph construction to retrieve information from point cloud data which we then feed into a dual-stream GNN to achieve the task of object classification.

1.1 Object Classification

Object classification is a deep learning process of determining the class or category to which an object stored in a file format belongs to. This is a task that is found in 2D

file format [8–10], where the pixels information are used to classify the image as well as in 3D file format such as the case of point clouds or mesh objects. Let us look at an example in Figure 1.1, that shows the classes determined from point cloud input data.

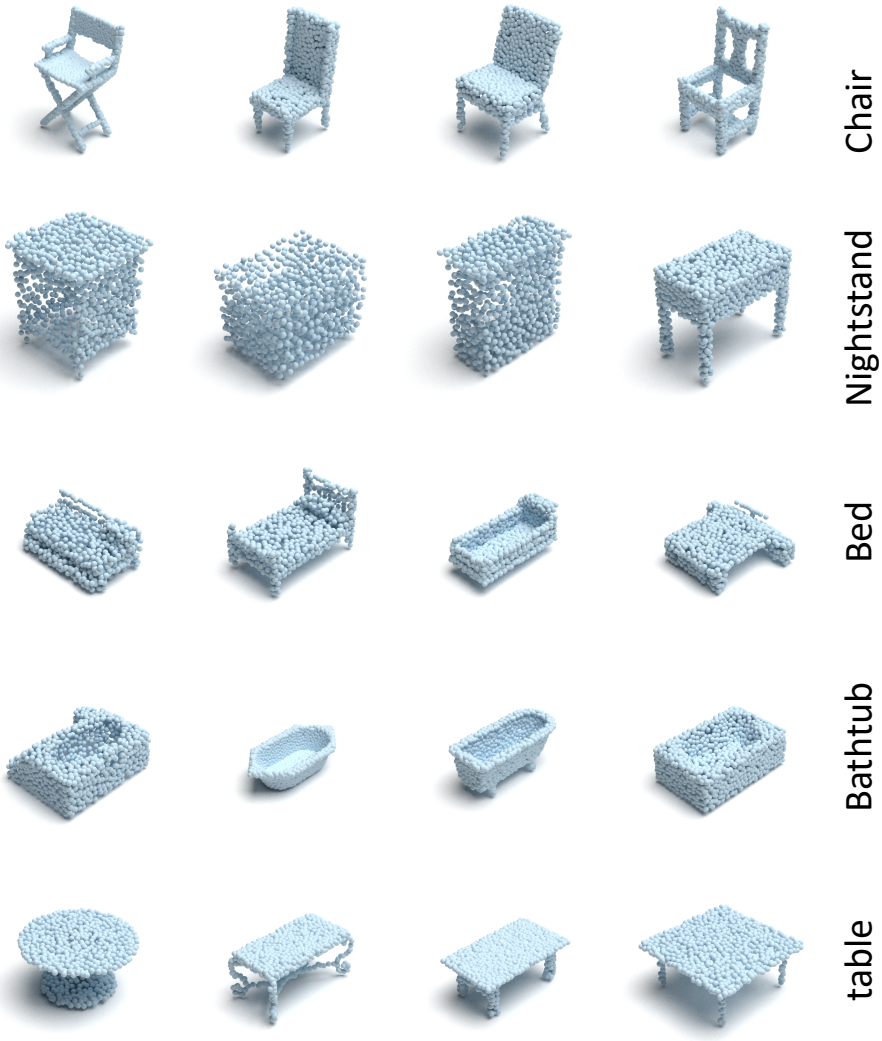


Figure 1.1: Example object classification on ModelNet40 [1]. We used PointVisualizaition [2] for visualizing these point clouds.

1.2 Point Cloud

Point cloud is a set of 3D points in space which represents an object, environment or surface. Each point p_i is stored using Cartesian coordinates (x, y, z). So, a point cloud dataset P containing N number of points can be denoted as $P = \{p_i | i = 1, 2, 3, ..N\}, p_i \in \mathbb{R}^3$. Optionally, points p_i can contain extra attributes such as colour or intensity. These points are obtained by technologies such as laser scanning, terrestrial scanning or photogrammetry, which capture the position and coordinates of each point relative to a reference system. Point cloud have been gaining popularity in many fields such as engineering, construction and robotics due to their ability to capture three-dimensional data easily with high accuracy and relatively smaller file size compared to other popular methods such as polygonal meshes.

Deep learning on 3D point cloud however does come with some challenges. This is due to a multitude of conditions and limitations of the current methods and technologies for point cloud gathering such as Lidar (Light Detection and Ranging). Below are some characteristics of point cloud data that makes extracting information from them challenging:

- **Irregularity:** Point cloud data does not have a consistent number of points per unit volume, leading to areas with dense number of points and other with sparse number of points. There is also occlusion that can happen due to obstruction in relation to the capturing device leading to surfaces not being sampled (Figure 1.2 (a)).
- **Unstructured:** Point cloud data only store the coordinate and optionally color of the surface that the point is representing, Thus information on how each individual point is connected to the neighboring points is missing. We cannot tell which set of points are on the same surface (Figure 1.2 (b)).
- **Unordered:** Compared to the grid arrangement of 2D images where pixels are stored in a known order, in point clouds data the points are stored in an unordered manner thus we do not have a known order to the points. This is particularly challenging for convolution-based methods since convolution requires

a known structured distribution of the inputs (Figure 1.2 (c)).

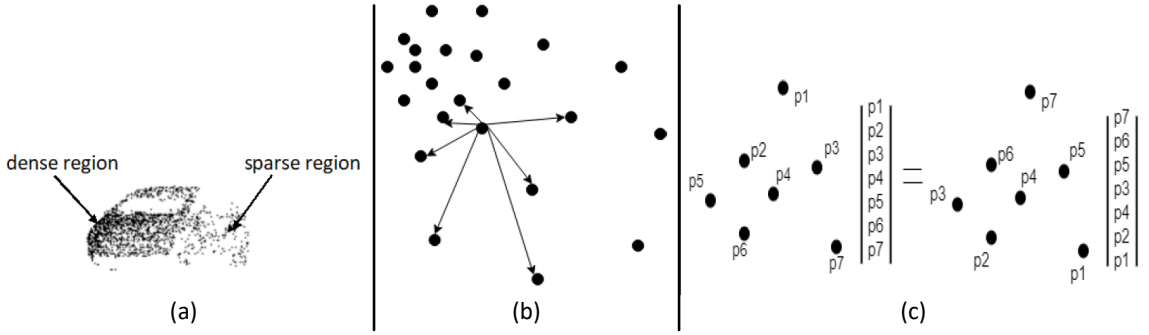


Figure 1.2: Challenges of point cloud data [3]. (a) Irregular: Sparse and dense regions. (b) Unstructured; No grid; each point is independent and the distance between neighboring points is not fixed. (c) Unordered: As a set, point clouds are invariant to permutation.

1.3 Disentangled Representation

The concept of disentangled representation is that data can be split into different regions based on some criteria and is closely linked with human reasoning (Bengio, Courville, and Vincent 2012 [12]). For example, Taihong et al. [13] separate facial images based on attributes such as smiling and not smiling and pair images of the opposite attributes to learn the particular attribute. In point cloud understanding, Xu et al. [4] proposed GDANet which uses Geometry-Disentangle Module. It uses a Laplacian operator on an adjacency matrix created in the feature space to create a high-pass filter. Using the high pass filter they calculate the l^2-norm of each points. Points with larger l^2-norm represents points with higher variations to their neighbors. Thus, they ordered the points in descending order based on the l^2-norm before retrieving the top and bottom M points to represent the sharp-variation component and gentle-variation component (see Figure 1.3).

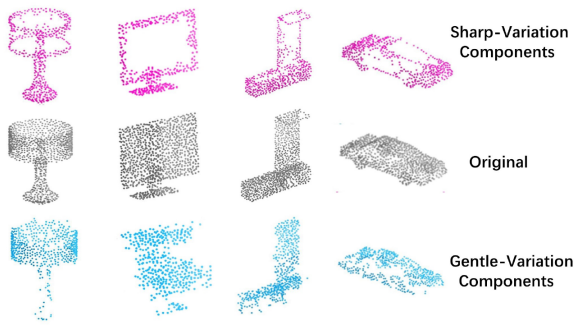
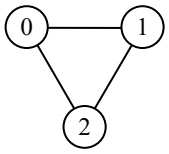


Figure 1.3: GDANet [4] geometry-disentangled point cloud objects

1.4 Graph

A graph is a pair $G = (V, E)$ where V is a set of vertices or nodes, and E is a set of edges and are represented by pairs (v_1, v_2) where v_1 and v_2 represents two vertices/nodes part of the set V. Graphs can be undirected or directed, in undirected graphs the edges (v_1, v_2) can be seen as being a connection both from v_1 to v_2 and from v_2 to v_1. See Figure 1.4 (a) below for a toy example of an undirected graph. However, in directed graph, the edges (v_1, v_2) represent a connection from a source node v_1 to a target node v_2.



(a)

0	1, 2
1	0, 2
2	0, 1

(b)

0	1	1
1	0	1
1	1	0

(c)

Figure 1.4: (a) Toy Graph example (b) adjacency list (c) adjacency matrix

Graphs are stored in different ways; more noticeably as adjacency list and adjacency matrix.

- **adjacency list:** In this approach, the graph is stored as a collection of unordered lists. The collection can be implemented as either an ordered list or a hashmap, where each list consists of vertices/nodes and represents the vertices/nodes to which the index or key vertex/node has an edge with (see figure 1.4 (b)).
- **adjacency matrix:** In this approach the graph is stored in a matrix of size $n \times n$ where n is the number of vertices/nodes in the graph (see Figure 1.4 (c)). Matrices are sometimes preferred due to the availability of efficient algorithm dealing with matrix operations. An unweighted and undirected graph represented by the adjacency matrix A can be defined as follows:

$$A_ij = \begin{cases} 1 : if\ there\ is\ an\ edge\ between\ v_i\ and\ v_j\ i.e.\ (v_i,\ v_j) \in E, \\ 0 : otherwise \end{cases}$$

1.5 Graph Neural Network (GNN)

GNN is a special type of neural network that operates on graph-structured data. In general, nodes of the graphs will contain features and the edges will represent relations between the nodes and may also contain features of their own. GNN have been successfully used to perform both for node-level [14–16] as well as graph-level classification [17–19]. There are different frameworks to GNN [20] such as Recurrent GNNs (RecGNNs) [21], Convolution GNNs (ConvGNNs) [4, 22, 23], Graph Autoencoders (GAEs) [24] and Spatial-temporal GNNs (STGNNs) [25]. We focus on ConvGNN framework that defines graph convolutions based on a node's spatial relations. Thus for each node we convolve its features with its neighbors' features to derive its updated features (see Figure 1.5). From another perspective it uses the idea of message-passing where information is propagated among the nodes along the edges of their neighbors. By applying multiple layers of graph convolution, each nodes attends to an increasingly greater region of the graph. Finally, we apply an aggregation function on the whole graph to extract graph-level classification.

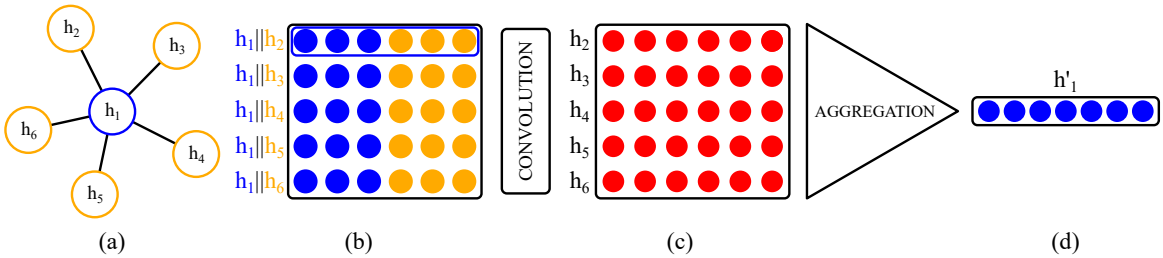


Figure 1.5: Graph Convolution pipeline for a node, h_1. (a) shows a toy graph example with h_1 as the central node and h_i neighbors. (b) concatenation of h_1 features with each neighbors features. (c) the resulting features after applying convolution on the features from (b). (d) the resulting new features h'_1 for node h_1 after applying an aggregation function on the features from (c).

1.6 Contributions

In this thesis, we analyze the use of a dual-stream GNN for point cloud classification. Our input are the gentle and sharp variation regions of an alpha complex graph representation of the point cloud. The sharp and gentle variation regions are obtained by applying either Gradient Structure Tensor or Corner and Edge geometric disentangled method. Our specific contributions can be summarized as follows:

- We propose the use of Gradient Structure Tensor, **GST** and Corner and Edge, **CE** to score the points based on their geometric space within the point cloud object, which we then use to split the object into gentle and sharp variation regions.
- We used alpha complex graph-based representation of the geometric split point cloud to extract the structural information of the point cloud object.
- We use a dual-stream GNN to combine the features extracted from the two alpha complex graph-based representations of the input point cloud for point cloud classification.
- We perform thorough experimental study on ModelNet40 [1] dataset to analyse the performance of the proposed network for point cloud classification.

1.7 Organization

This thesis is organized as follows:

- **Chapter 1 (Introduction)** introduces the studied problem, explains the key challenges, motivation and contributions.
- **Chapter 2 (Related Work)** provides a summary of other related studies that proposes methods and approaches for 3D point cloud object classification.
- **Chapter 3 (Method)** covers the graph generation and neural network proposed in this thesis for point cloud classification.
- **Chapter 4 (Experimental Results)** demonstrates the results of the proposed method for point cloud classification on ModelNet40 [1] dataset.
- **Chapter 5 (Conclusion and Future Work)** discuss the conclusion we drew from the experimental results and proposes the future directions we saw to improve the proposed model.

Chapter 2

Related Work

2.1 3D Object Classification

3D Object Classification consist of determining the category (e.g: chair, table, car, ...) to which an object belongs. With the advancement of hardware and availability of publicly datasets from industries and universities such as ModelNet40 [1], ModelNet10 [1], ModelNet-C [26], Sydney Urban Objects [27], ScanNet [28], ScanObjectNN [11], interest in academic research within point cloud classification has gained much interest. The following sections will cover different methods that have been developed in recent years.

2.2 Multi-view-based Methods

Multi-view-based methods takes advantage of the advancement made in 2D CNNs by projecting the 3D shape into multiple views (see Figure 2.1), then extracting view-wise features before aggregating them into a single global descriptor.

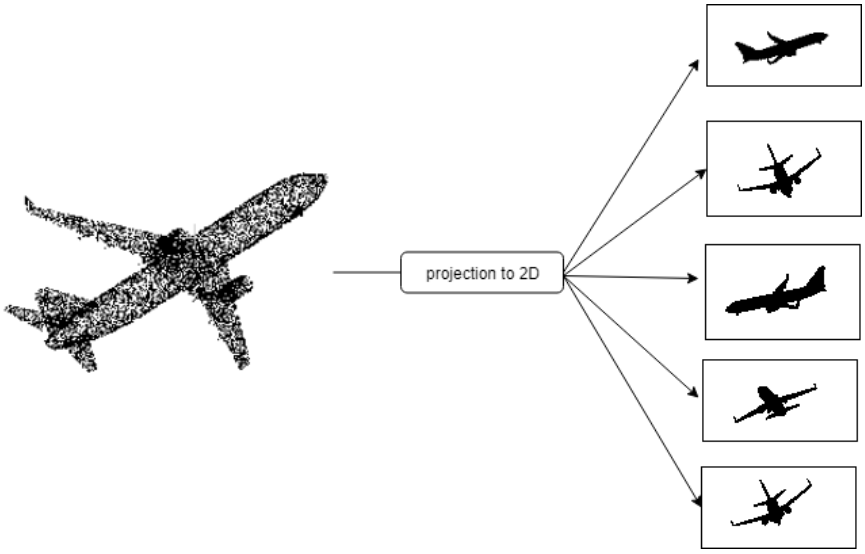


Figure 2.1: Multi-view projection of a 3D point cloud object into 2D images. Each 2D image represents the same object viewed from a different angle. [3]

The pioneering work for this approach was proposed in 2015 by Su et al. [29] the Multi-View Convolutional Neural Network (MVCNN). In this method a set of known viewpoints are always used to extract the different views of the object. The features extracted from the different views are then max-pooled into a single global descriptor for the object. Multi-resolution filtering extension, which captures information at multiple scales, was introduced by Qi et al. [30]; besides, the authors used data augmentation for better generalization to improve on MVCNN [29]. RotationNet [31] improves on MVCNN [29] by considering the viewpoints labels as latent variables which are learned in an unsupervised manner during training, thus it only uses a subset of the views which is further helpful in practical scenarios when only partial views of the object is available.

2.3 Volumetric-based Methods

These methods usually try to emulate the structured behaviour found in images by applying methods to voxelize the points into 3D grids (see Figure 2.2), then applying 3D Convolution Neural Network (CNN) on the volumetric representation for shape classification.

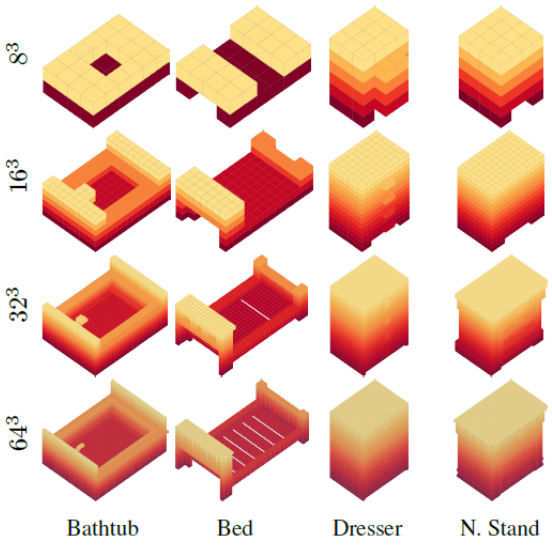


Figure 2.2: Voxelized 3D Shapes from ModelNet10 as presented in OctNet paper [5]

Maturana et al. [32] introduced the VoxNet which voxelize the input point cloud into $I \times J \times K$ voxels, then depending on the occupancy model used each voxel is represented as a single feature which is then preprocessed before being feed into a CNN for classification. Wu et al. [1] proposed a convolutional deep belief-based 3D ShapeNets to learn the distribution of points from various 3D shapes. The issue with such methods was the computation and memory cost thus Gernot et al. [5] proposed the OctNet which partitioned the point cloud using a hybrid grid-octree structure (see Figure 2.2), which represents the scene with several shallow octrees along a regular grid. The structure of the octree is encoded efficiently using a bit representation and the feature vector of each voxel is indexed by simple arithmetic. Le et al. [33]

proposed PointGrid which normalizes the point cloud to unit boxes, then employs Point Quantization- a sampling method that ensures each voxel has exactly K points. Thus they are able to share 3D Convolution Kernel to create feature maps for each voxel.

2.4 Point-based Methods

Point-based methods processes the points directly, without converting them into any structures such as in Volumetric-based Methods or any other form such as in Multi-view-based Methods. The pioneering work for this approach was proposed in 2017 by Qi et al PointNet [34] which used several MLP layers and extracted the global features with a max-pooling layer.

PointNet [34] learned features from each points independently, thus losing any local structure information between points. To alleviate such issues, Qi et al. [35] proposed a hierarchical network PointNet++ to capture fine geometric structures from the neighborhood of each point. Zhao et al. proposed PointWeb [36] which utilizes the context of the local neighborhood to improve point features using Adaptive Feature Adjustment (AFA). Point Attention Transformers (PATs) [37] uses Absolute and Relative Position Embedding (ARPE) module to represent each point into a high-level representation, then the features pass through layers of Group Shuffle Attention (GSA) block and down-sampling blocks, either Furthest Point Sampling (FPS) or Gumbel Subset Sampling (GSS) before being connected to an MLP for classification.

2.5 Graph-based Methods

Graph-based approaches extract information from the point cloud by representing it using graph structures. In general, the points represent the nodes in the graph and edges are created between the nodes based on an algorithm, then using a graph convolution neural network the features from the nodes are learned to achieve tasks such as classification.

Klokov et al. [38] proposed the use of kd-tree, which is a special kind of graph. The kd-tree is built in a top-down manner on the point clouds to create a feed-forward kd-network with learnable parameters in each layer. The computation performed in the kd-network is in a bottom-up fashion. The leaves represent the input points; two nearest-neighbor (left and right) nodes are used to compute their parent node using the shared parameters of a weight matrix and bias. Wang et al. [22] proposed DGCNN, which uses edge convolution, edgeConv. In edgeConv, for each point the k-nearest points are gathered, then the coordinates of the central node and the difference between the coordinates of the neighbors and the central node are concatenated. These features then goes through a layer of convolution to extract the new features for the points/nodes. Between each previously detailed edgeConv layer, the k-nearest neighbor (k-NN) graph is reconstructed using the new features, After the last edgeConv layer the features goes through a global max-pooling operation similar to PointNet [34] finally classification is achieved by applying MLP over the extracted features. In 2021 Wang et al. [39] proposed the use of deep normalized Reeb graph convolution. This paper used the generalised Reeb graph construction on point clouds, together with k-NN graph to extract information from the point cloud for classification. Xu et al. [4] proposed the GDANet, which improved on DGCNN [22] by using Geometry-Disentangle Module which splits the object features into gentle-variation and sharp-variation regions, which are learned independently in an unsupervised manner by the network. Srivastava et al. [23] proposed GeomGCNN, which builds on DGCNN [22] proposing two main improvements, to augment the vertex representations with important local geometric information of the points and an improvement to constructing k-NN graph taking into consideration sampling frequency difference, these improvements allowed them to create the model which at the time of writing this thesis holds the best accuracy on ModelNet40 [1] classification task.

2.6 Transformer-based Methods

Following unprecedented success in natural language processing (NLP), Transformers have been gaining much attention lately. This increase in interest and success at-

tributed to Transformers' impressive ability to model long-range dependencies have not only made it a very compelling option for NLP but also for computer vision tasks. In general, Transformers work by creating tokens from subpart of the inputs and finding the relationship between them to make sense of the overall input.

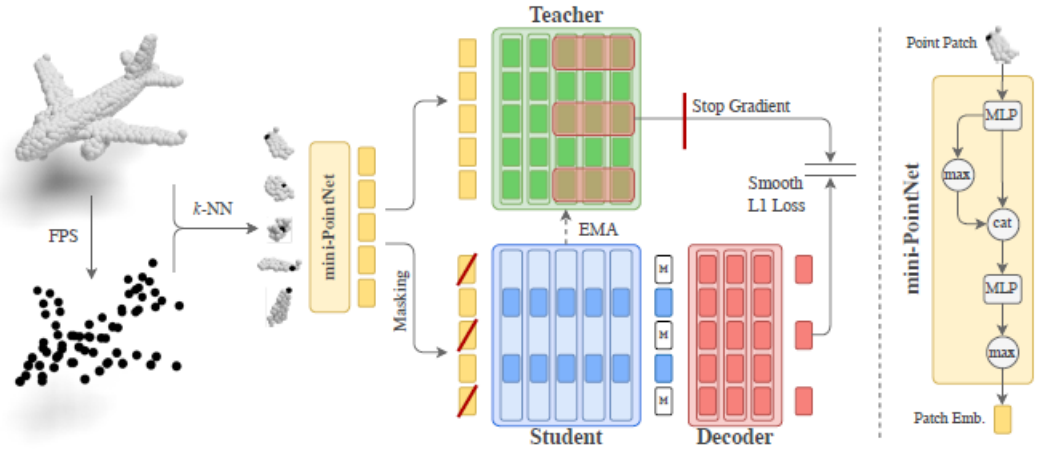


Figure 2.3: Point2Vec [6] pipeline

For example, in NLP, the words are embedded and the relationship between the tokens in the phrase allow the model to understand the context and meaning of the phrase. Dosovitskiy et al. [40] created a 2D image classification network using transformers by splitting the image into $k \times k$ parts, then embedded them together with a positional embedding to classify the image. In 2022, Yu et al. [41] proposed Point-BERT inspired by the success of BERT [42] (Bidirectional Encoder Representations from Transformers) a language representation model. They pre-trained the point cloud Transformers by dividing the point cloud into several local point patches and using a point cloud Tokenizer with a discrete Variational AutoEncoder (dVAE) they obtained tokens from the patches. Then they randomly masked out some patches of the input point clouds and fed them into the backbone Transformers. The pre-training objective was to recover the original point tokens at the masked locations under the supervision of point tokens obtained by the Tokenizer. Using these learned tokens together with a classification head made up of a MLP they were able to achieve

classification task in point cloud. Zeid et al. [6] proposed the Point2Vec which was inspired by the ability of extending Data2Vec [43] for 3D point clouds. In Data2Vec, they proposed a student–teacher pre-training framework for the creation of tokens. Point2Vec extended that framework for 3D point cloud and found that one of the issues that it faced was that the leakage of positional information revealed the overall object shape to the student even under heavy masking, which hampered data2Vec ability in learning strong representations for point clouds. They fixed the issue by adopting an approach inspired by MAE [44]. They only fed the non-masked embeddings to the student. A separate decoder, implemented as a shallow Transformer encoder, took the output of the student and the previously held-back masked embeddings as input and predicted the training targets. Figure 2.3 shows the stages involved in Point2Vec architecture.

2.7 Summary

The current direction in point cloud processing focuses on machine learning. Thus it has to deal with the issues that all the current machine learning approaches face. Among these drawbacks are memory space and inference time. Both these issues are usually interrelated; a deep neural network will have a lot of weights consuming large memory space and a lot of computations are required to learn from the inputs as they pass through the network thus resulting in huge processing time.

Transformers which have been gaining a lot of interest in computer vision generally results in high accuracy. However, Transformers tends to be large in size and thus incurs heavy computational time. On the other hand, we can prioritise inference speed by using point-based learning but the accuracy achieved using point cloud directly tends to be lower. Graph-based methods tends to have a good trade off between accuracy and speed. Thus we chose to use a graph-based method. We attempt to create a graph that would reduce the number of inputs (edges in the case of a graph) such that it would prioritize those that would provide more important details. Thus we would be able to maintain high accuracy while reducing the number of edges and therefore memory space.

Chapter 3

Method

This chapter presents different components of our model's pipeline. The first stage in our pipeline deals with the graph construction that takes input point cloud and generates an alpha complex (section 3.1). The next stage focuses on splitting the object into regions of gentle and sharp variations (section 3.2). This is followed by feature extraction and learning on the disentangled alpha complex via graph neural network (section 3.3). See Figure 3.1 for an outline of our model.

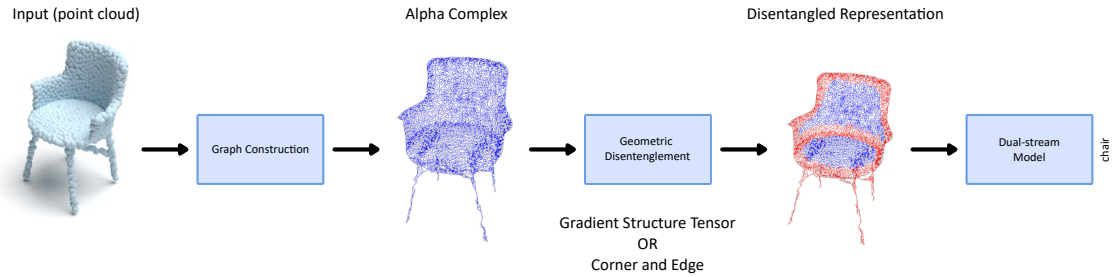


Figure 3.1: Pipeline of our model.

3.1 Graph Construction

Let $P = \{p_i | i = 1, 2, 3, ...N\}, p_i \in \mathbb{R}^F$ where F is the set of features associated to each points $F = \{f_i | i = 1, 2, 3, ...n\}$. In point clouds, the features are usually the coordinates of the points followed by additional features such as normal, reflectance or RGB-color. In the case of ModelNet40 [1] dataset, points are provided with their spatial coordinates. Thus each point p_i can be denoted as $p_i = (x_i, y_i, z_i)$ where x_i, y_i, z_i are the corresponding axis coordinate of the point in Euclidean space.

By considering each point in the point cloud as a vertex, we construct an alpha complex graph over the whole point cloud. Since alpha complex is a subcomplex of the Delaunay graph over a set of points we first need to define the Delaunay graph.

To facilitate the understanding of Delaunay graph, we present the explanation over a set of 2D points, and then we use that intuition to expand into higher dimensions as detailed by Maur et al. [45]. Delaunay graph over a set of points in 2D space can be defined as the triangulation of the points such that no point lies within the circumcircle (see Figure 3.2) of a triangle. See Figure 3.3 for example of Delaunay graph on points in 2D space.

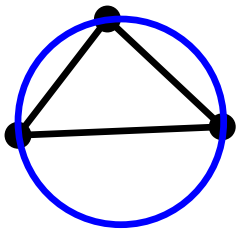
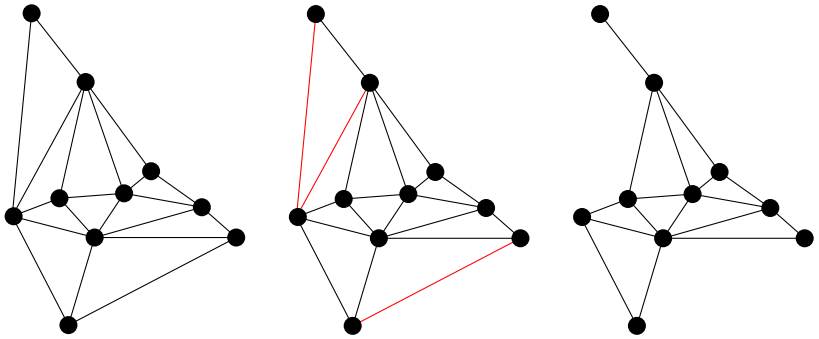


Figure 3.2: Circumcircle of a triangle.

(a) Delaunay graph. (b) Non Delaunay graph.

Figure 3.3: Illustration of Delaunay and non-Delaunay graph. (a) the triangulation satisfy the condition of Delaunay graph. (b) the triangulation does not satisfy the conditions since the point in red lies within the circumcircle in red of another triangle.

Using the graph obtained by applying Delaunay triangulation, we can apply the alpha complex filtration in which we remove all edges that have an Euclidean distance greater than the α-value. See Algorithm 1 for the pseudo code we used to implement Alpha Complex graph for a point cloud, p with α-value, α. (see Figure 3.4 below for alpha complex example from Delaunay triangulation).



(a) (b) (c)

Figure 3.4: Creating alpha complex from Delaunay Triangulation. (a) Delaunay Triangulation, (b) red edges greater than the α-value, (c) Alpha complex

Algorithm 1 Alpha Complex

```
1: procedure ALPHA COMPLEX(p, α)
2:     Edges ← Delaunay(p)
3:     for edge in Edges do
4:         if ‖edge‖ > α then
5:             Remove edge from Edges
6:         end if
7:     end for
8:     return Edges
9: end procedure
```

We decided to use alpha complex in order to create graphs that models the shape of the object. As we can see the Delaunay graph construction (Figure 3.5 (a)) creates an object that contains too many edges and do not reflect the underlying shape of the object. Using alpha complex filtration we are able to remove the extra edges (Figure 3.5 (b)), however we need to choose the right α-value in order to remove the correct amount of edges to reflect the shape of the object as best as we can. (Figure 3.5 (c) shows the effect of choosing an α-value that is too small, resulting in too many edges being removed).

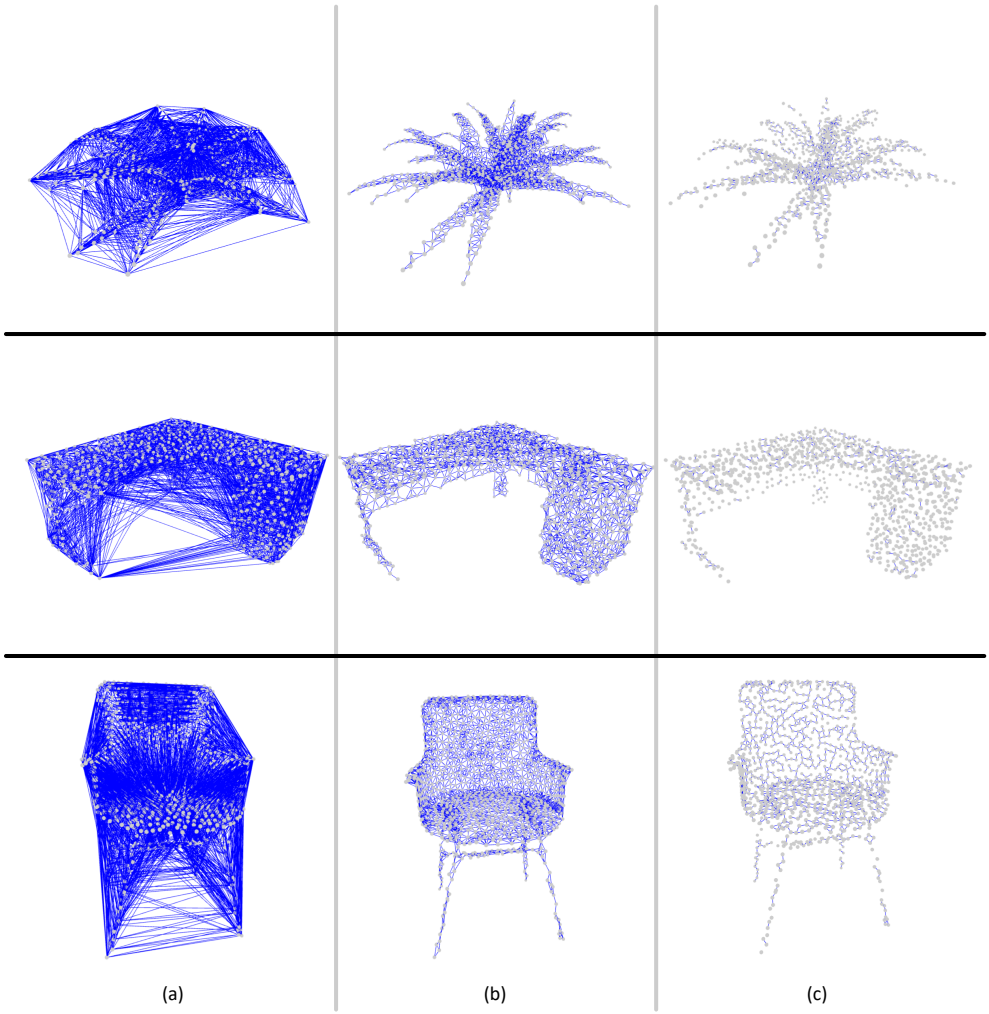


Figure 3.5: (a) Delaunay Graph (b) Alpha complex α-value=0.04 (c) Alpha complex α-value=0.02.

3.2 Geometric Disentangled Representation

Geometric Disentangled Representation for point cloud can be thought of as splitting the object into regions of gentle and sharp-variations similar to that proposed in GDANet [4]. Regions of gentle-variations refer to the regions with flat surfaces, while regions of sharp-variations refer to corner and edges. In this thesis, we considered

two methods for deriving geometric disentangled representation based on the same intuitive idea as in GDANet [4], namely a simplified implementation of Gradient Structure Tensor proposed by Chen et al. [46] (we will refer to this method as **GST** throughout this thesis) and a simplified version of the corner and edge detection method (we will refer to this method as **CE** throughout this thesis) proposed by Ahmed et al. [7]. The following subsections covers the simplified implementations of these two methods. In both methods we derive a continuous score which we call confidence similar to how it is used in GST [46]. After calculating the confidence score for all the points within the point cloud, we sort them in descending order and pick the top M points as points belonging to the sharp-variation region and bottom M points as points belonging to the gentle-variation regions.

3.2.1 Gradient Structure Tensor (GST)

In our work the confidence score C^i of each point p_i is calculated based on the neighboring points within a sphere centered at p_i. To improve robustness of our classification, spheres of various radii are used to gather the neighboring points of p_i. For each neighboring sphere, Principal Component Analysis (PCA) [47] is applied to the neighboring points in order to obtain the variance of the three main components, $\lambda_0^i, \lambda_1^i, \lambda_2^i$ where $\lambda_0^i \leq \lambda_1^i \leq \lambda_2^i$. Using these components the two geometric properties namely **fitting quality** C_f and **sampling uniformity** C_s are calculated.

Fitting quality C_f represents the fitting quality of the local tangent plane at point p_i, calculated using Equation 3.1.

$$C_f^i = \lambda_0^i / (\lambda_0^i + \lambda_1^i + \lambda_2^i) \tag{3.1}$$

If point p_i and its local neighbors can perfectly fit the local tangent plane, the value of the fitting quality measure C_f^i of p_i approaches 0. In contrast, if the neighbors of point p_i are inhomogeneous, they are most probably distributed on edges, corners and any extrusions/intrusions of the shape. In this case, the value of C_f^i of p_i tends to be 1.

Sampling uniformity C_s represents the local sampling uniformity, as quantified by Equation 3.2.

$$C_s^i = \lambda_1^i / \lambda_2^i \tag{3.2}$$

If point p_i and its local neighbors are distributed linearly, the value of C_s of p_i approaches 0; if uniformly distributed, Cs tends to be 1; therefore, this measure is effective in detecting outer boundaries of the shape.

These two geometric properties are then combined to get the complete confidence score $C^i \in [0, 1]$ of point p_i (Equation 3.3).

$$C^i = 1 - \frac{1}{n} \sum_{j=1}^{n} (1 - 3C_f^i) \cdot C_s^i \tag{3.3}$$

where n represents the number of neighboring spheres used. Figure 3.6 shows the color map and region split between points that fall under gentle-variation region and sharp-variation region. We prefer the use of the parameters within the (b) column as it deals better with differences in density across the point cloud relative to parameters in column (a), while also being able to score finer details relative to parameters in column (c).

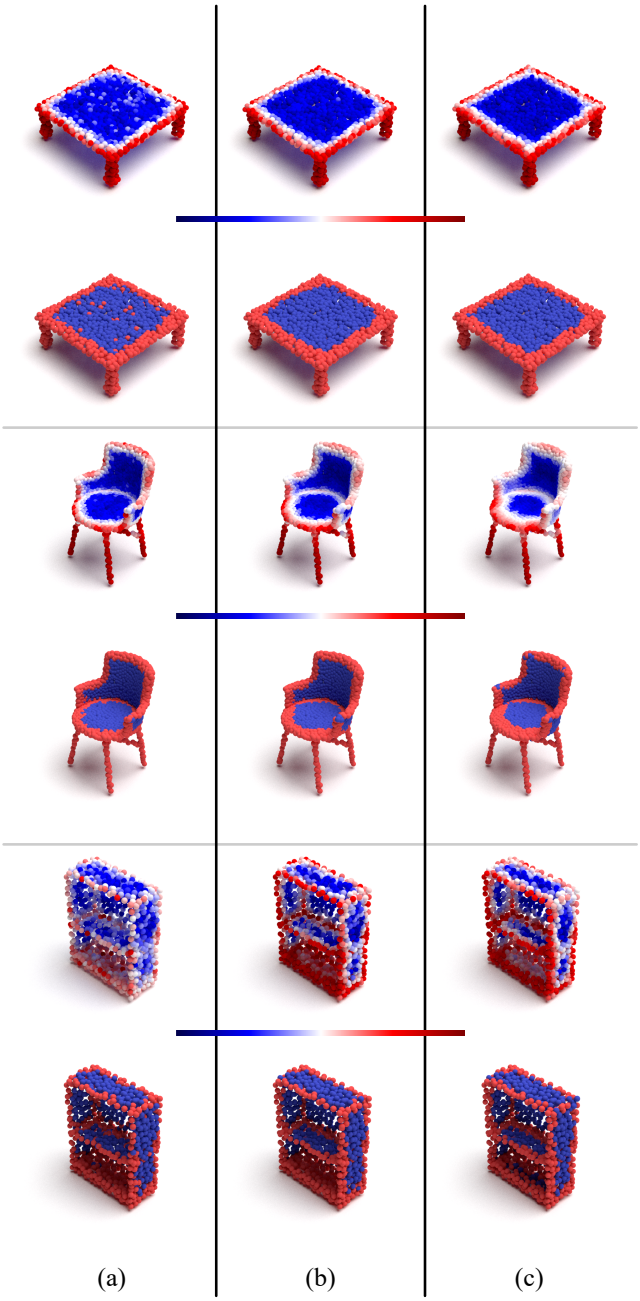


Figure 3.6: A few example showing the classification of points into regions of gentle and sharp variations using GST method.Top row of each section represents the color map of the confidence score for each point with the normalised color map range under the color map. Bottom row shows the region to which the point belongs to, blue for gentle-variation and red for sharp-variation. (a) sphere radius=[0.1, 0.15, 0.2] (b) sphere radius=[0.15, 0.2, 0.25] (c) sphere radius=[0.2, 0.25, 0.3].

3.2.2 Corner and Edge (CE)

This method is originally proposed in the paper by Ahmed et al. [7] and is aimed at detecting corners and edges for robotic welding. In this thesis, we utilise the continuous nature of the simple yet effective portion of their method to derive a way to create a similar idea as confidence as described in GST [46].

The algorithm proposed in the CE [7] paper is based on an intuitive reasoning that the mean of a patch of points taken from a specific query point should be closer to the query point if it is on a flat surface and further if it is at an edge or corner(see Figure 3.7).

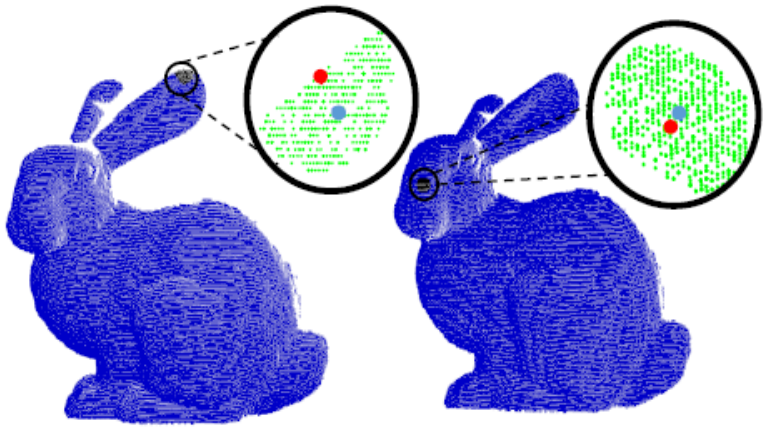


Figure 3.7: Bunny point cloud example from Ahmed et al. [7] paper. Points in green refers to the k-nn points based on the query point in red. The point in blue within the magnified circle refers to the resulting point derived from calculating the mean of the k-nn points

To calculate the confidence score C^i for each query point $p_q^i \in P$ where P refers to the set of all points in the point cloud. We calculate the Euclidean distance between the mean point p_m^i obtained by taking the mean of the k-nn points from the query point p_q^i (Equation 3.4) and the query point p_q^i itself (Equation 3.5).

$$p_m^i = \frac{1}{N} \sum_{j \in \mathcal{N}_i} p_j \tag{3.4}$$

N refers to the number of points within k-nn, and $\mathcal{N}_i$ refers to the neighbors of point p^i.

$$C^i = d(p_m^i, p_q^i) = \|p_m^i - p_q^i\| \tag{3.5}$$

Figure 3.8 shows the color map and region split between points that falls under gentle-variation region and sharp-variation region. It can be observed that the results of Fig. 3.8 (b) deals better with differences in density across the point cloud relative to Fig. 3.8 (a), while also being able to score finer details relative to Fig. 3.8 (c).

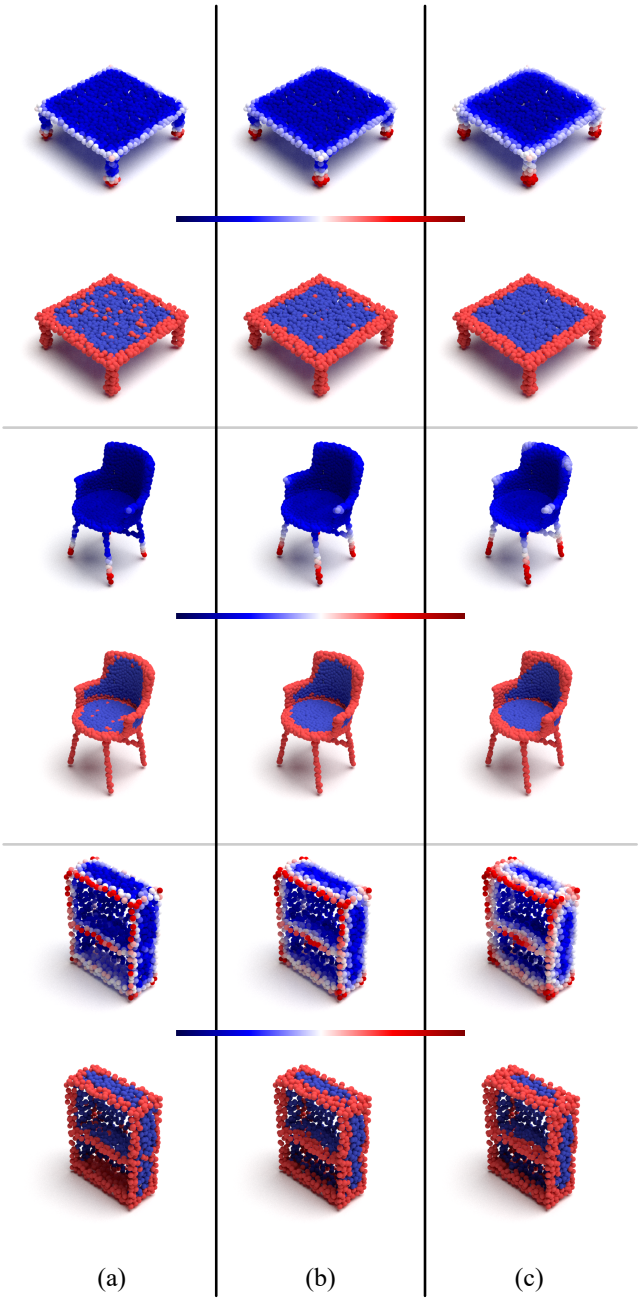


Figure 3.8: A few example showing the classification of points into regions of gentle and sharp variations using CE method. Top row of each section represents the color map of the confidence score for each point with the normalised color map range under the color map. Bottom row shows the region to which the point belongs to, blue for gentle-variation and red for sharp-variation. (a) k-nn=32 (b) k-nn=64 (c) k-nn=128.

3.3 Graph Neural Network for Classification

In this thesis, we use a Graph Attention Network (GAT) for our downstream task, i.e., point cloud classification based on the graph representation of the point cloud object. GAT was originally proposed by veličković et al. [15] and later improved by brody et al. [48]. Below, we describe the mathematical underpinnings of GATv2 [48] layer for a specific node v_i with features h_i.

First, we concatenate (Equation 3.6) the features of the query node h_i with that of its neighbor, $h_j \in \mathcal{N}_i$, where $\mathcal{N}_i$ represents the set of neighbor nodes of node v_i, for the first layer where the node features are the coordinates of the points. We used $h_j = h_j - h_i$ in our work to untangle the positional and structural information:

$$h_i || h_j \tag{3.6}$$

Then we multiply (Equation 3.7) the concatenated features with a learnable weight matrix $\mathbf{W} \in \mathbb{R}^{\mathbf{F}' \times \mathbf{F}}$ where F is the number of the feature in $h_i || h_j$ and F' is the number of feature for the output dimension of that layer:

$$\mathbf{W} \cdot [\mathbf{h_i} || \mathbf{h_j}] \tag{3.7}$$

We then apply an activation function σ to the result (Equation 3.8), the most common σ used here is $LeakyReLU$:

$$\sigma(\mathbf{W} \cdot [\mathbf{h_i} || \mathbf{h_j}]) \tag{3.8}$$

In case of Graph Convolution Neural Network, the results from the above equation is aggregated (using max pooling, summation, average, etc..). However, in GATv2 it is multiplied by the corresponding attention mechanism coefficient α_{ij}^k (Equation 3.9). The attention mechanism coefficient is learned using a feed forward layer with a $Softmax$ activation function.

$$\alpha_{ij} \cdot \sigma(\mathbf{W} \cdot [\mathbf{h_i}||\mathbf{h_j}]) \tag{3.9}$$

In order to get the features for the node h_i after that layer, we aggregate the features of the neighbors as in Equation 3.10.

$$\sum_{j \in \mathcal{N}_i} \alpha_{ij} \cdot \sigma(\mathbf{W} \cdot [h_i||h_j]) \tag{3.10}$$

We also apply multi-head attention with K heads, which means that we follow the Equation 3.10 K times and concatenate the results to obtain the new feature for h_i as presented in Equation 3.11.

$$h_i' = \Big\Vert_{k=1}^{K} \left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \cdot \sigma(\mathbf{W}^k \cdot [h_i||h_j]) \right) \tag{3.11}$$

3.3.1 Single-Stream Architecture

In this section, we present the proposed architecture for our single-stream model using GATv2 layers (see Figure 3.9). The structure of the proposed model is inspired by DGCNN [22].

Spatial transform module (bottom diagram in Figure 3.9 taken from DGCNN [22]) used to deal with the discrepancies in rigid transformations such as rotation, translation and scaling of the input point clouds.

Following the spatial transform module, we used a set of four GATv2 layers with output dimensions 64, 64, 64 and 128. The outputs of these layers are then concatenated and goes through a layer of convolution with output dimension 1024. Finally, we use a multi-layer perceptron, MLP (dimensions 512, 256, C) where C is the number of categories we are classifying.

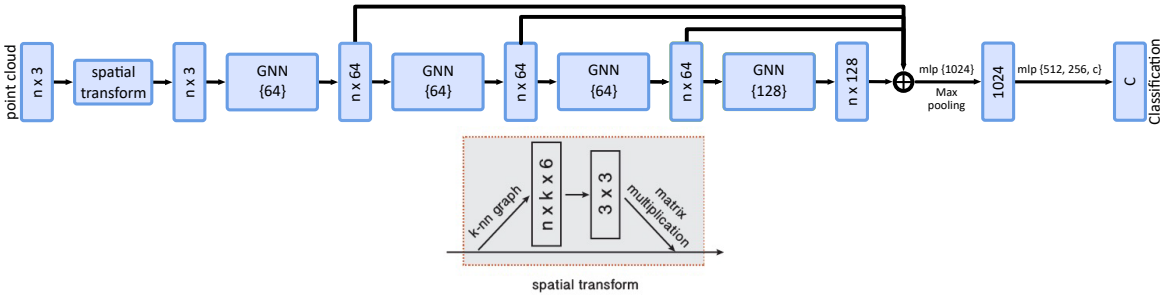


Figure 3.9: Single-stream GNN architecture. Spatial transform module diagram.

3.3.2 Dual-Stream Architecture

In our dual-stream architecture (see Figure 3.10), we start by ordering the features of the points based on their confidence score obtained by using either GST (section 3.2.1) or CE (section 3.2.2) as discussed in the corresponding sections. We then split the set of point features into two equal parts, that we feed into separate GNN layers. The results of these layers are then stacked together to be fed into a single stream GNN layer, we repeat this process along our model before using a 2D Convolution layer with skip connections to previous layers, followed by MLPs which ends with the classification output layer.

The splitting of the GNN layer into two allows the model to train the weights to better attend or fit the type of features most common to gentle and sharp variation regions separately. It is then combined to ensure that the model does not over fit the features and learns the global features of the whole shape.

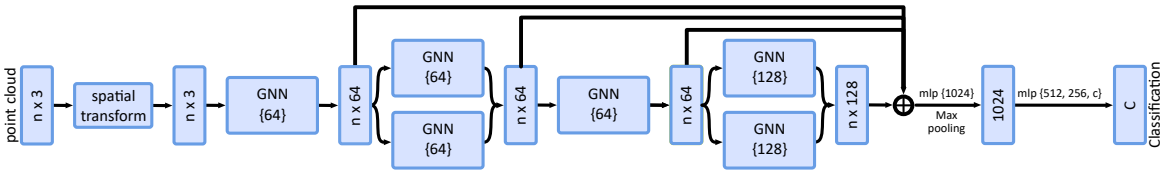


Figure 3.10: Dual-stream GNN architecture.

3.4 Model Training Configurations

For both models presented in the previous sections (Section 3.3.1 and 3.3.2) we used the following configurations during the training phase. We used softmax cross entropy for our loss function, Adam [49] as our optimizer with an exponential decay learning rate initially at 0.001 with a decay rate of 0.7 every 200000 steps. We trained for 400 epochs with a batch size of 32.

Chapter 4

Experimental Results

4.1 Implementation and Training

We implemented our models based on the code available from DGCNN GitHub page [50]. We used Python 3.7.7, Numpy [51], Scikit Learn [52], Scipy [53] and Tensorflow [54]. We also used Open3d [55] and PointVisualizaiton library [2] for point cloud visualization and Matplotlib [56] for graph plots throughout this thesis. We trained our model on Compute Canada's Béluga cluster and our models made use of a single NVIDIA Tesla V100-SXM2 GPU with 16GB memory.

4.2 Dataset

We evaluated the classification accuracy of our models on ModelNet40 [1]. Modelnet40 benchmark dataset contains 12,311 pre-aligned shapes from 40 categories, which are split into 9,843 (80%) for training and 2,468 (20%) for testing. A sample of ModelNet40 point cloud objects and corresponding labels are shown in Figure 1.1.

4.3 Results

In this section, we compare the performance of our model with other state-of-the-art 3D object classification models and provide a more in-depth analysis of our model's inner layers and performance.

4.3.1 Comparison

We measure and compare the performance of our model based on two commonly used metrics in object classification namely **overall accuracy** and **mean accuracy** abbreviated to **OA** and **mAcc** respectively.

- **Overall Accuracy (OA)**: OA refers to the overall probability of our model classifying all the objects it is tested on. In other words, it is the probability of our model being successful if used in a setting where there is an equal chance of having to classify each category an equal amount of time. Equation 4.1, shows the formula used to calculate overall accuracy.

$$\text{OA} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Items}} \tag{4.1}$$

- **Mean Accuracy (mAcc)**: This metric quantifies the average probability of our model classifying each individual category. In other words, it is the probability of our model being successful if used in a setting where there is a specific set of categories to be classified. Equation 4.2, shows the formula used to calculate overall accuracy.

$$\text{mAcc} = \frac{\sum \text{Accuracy for Each Class Prediction}}{\text{Number of Classes}} \tag{4.2}$$

These two metrics are good descriptors of the performance of object classification models and by comparing them we can tell if a model is doing particularly bad in specific categories, i.e if the difference between OA and mAcc is large we can tell that there must be specific categories that are some what outliers and performing

considerably worst relative to the other categories.

Method	OA	mAcc
PointNet [34]	89.2	86.0
PointNet++ [35]	90.7	-
DGCNN [22]	92.9	90.2
GDANet [4]	93.8	-
Kd-Net [38]	90.6	86.3
OctNet [5]	86.5	83.8
DNRG [39]	89.9	87.1
GeomGCNN [23]	95.9	93.1
Ours (alpha)	88.3	84.2
Ours (alpha + knn)	91.5	88.1
Ours (knn)	90.7	87.1
Ours (Dynamic knn)	90.6	87.5

Table 4.1: Classification accuracy on ModelNet40 [1]. All accuracies in %. The results for our models are taken from the dual-stream models presented in Section 4.4.2 for Ours (alpha) and Section 4.4.5 for the other graph constructions.

Table 4.1 shows the comparison between our proposed approach and other existing methods in classifying the 40 classes in the ModelNet40 [1] dataset. The accuracies of other methods reported in Table 4.1 are extracted from the respective papers. Our graph based method performs better than some other graph based methods such as OctNet [5] and Kd-Net [38] as well as other point-based methods such as PointNet [34] and PointNet++ [35]. The proposed model did not however perform better than newer state-of-the-art models such as DGCNN [22], GDANet [4] and the current best method as far as we are aware at time of writing this thesis GeomGCNN [23]. We suspect that our lower accuracy might have to do with the need to have a larger number of edges to describe the local feature in greater details since we can see from section 4.4.2 that we are able to gain considerable improvements by having extra edges that describes the local feature at each point. We are also prone to difference

in point density a key issue that GeomGCNN [23] proposed a solution for in order to get better accuracy.

4.3.2 Model Analysis

In this section, we take a more in depth look at what our model is doing within the hidden layers by analyzing the feature spaces within the inner layers of our dual-stream model, presented in Figure 4.1 and we analyze how well our model is distinguishing between the different classes by taking a look at the classwise accuracies presented in Table 4.2 and at the t-SNE plot presented in Figure 4.2.

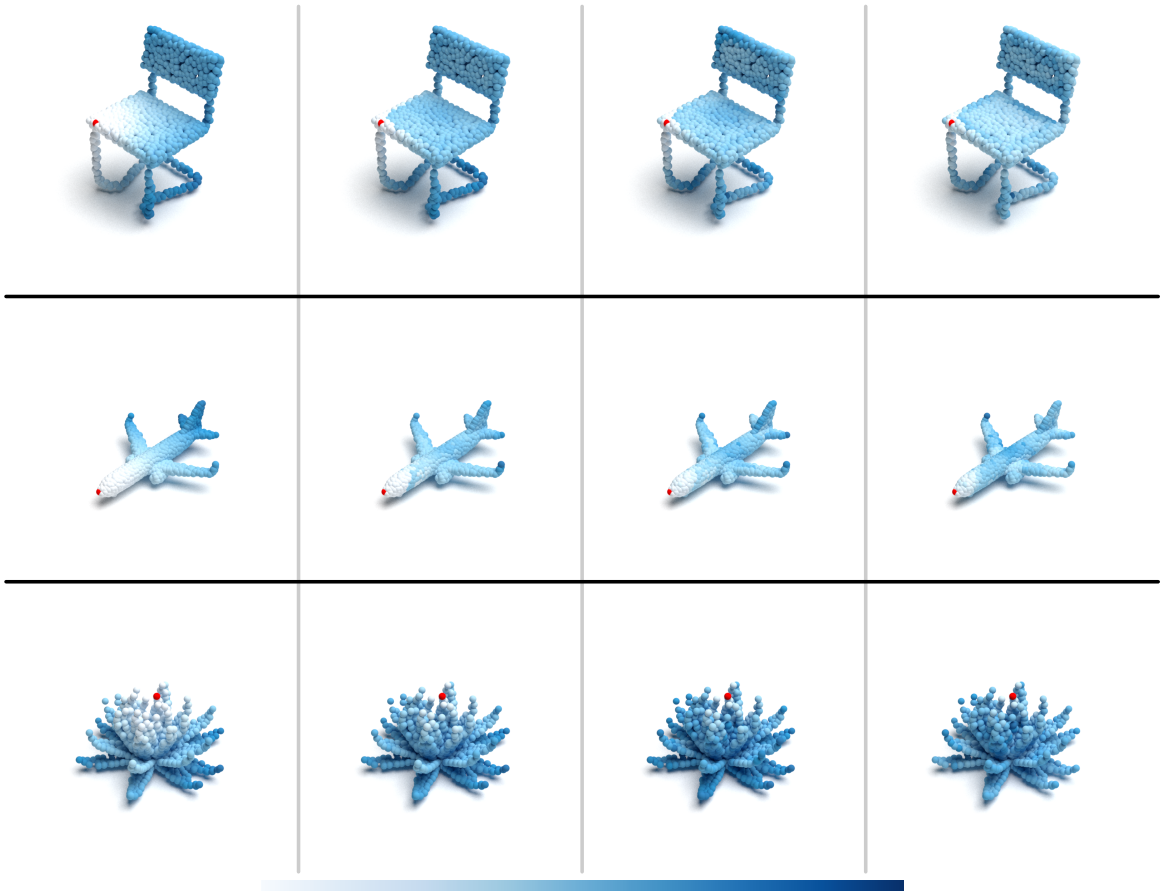


Figure 4.1: Feature spaces generated after the different layers of the model. Feature attention based on the red point.

Figure. 4.1 visualizes the feature attention between the point in red and the relative neighboring points. Going from left to right i.e from layer 1 to layer 4 of our model, the feature being attended to based on the red point become more coarse and focuses on the feature itself such as the corner or edge in case of the chair or the rounded surface of the airplane nose in case of the airplane. We can see especially in the case of the chair that the geometric disentanglement that split the object into two in the second layer made the model focus on the outer surface of the chair more than the flat surface in relation to the query point in red.

Class	Acc	Class	Acc	Class	Acc	Class	Acc
airplane	100	cup	65	laptop	100	sofa	97
bathtub	92	curtain	85	mantel	94.9	stairs	85
bed	99	desk	93	monitor	98	stool	80
bench	75	door	95	night_stand	79.1	table	81
bookshelf	97	dresser	84.9	person	90	tent	95
bottle	97	flower_pot	10	piano	93.9	toilet	97
bowl	100	glass_box	97	plant	82	tv_stand	88
car	99	guitar	99	radio	60	vase	84.8
chair	98	keyboard	100	range_hood	94	wardrobe	70
cone	95	lamp	95	sink	95	xbox	85

Table 4.2: Classwise accuracies for alpha+knn model. All accuracies in %

We note from Table 4.2 that our model performs generally good in most classes, however, classes such as flower pot, cup and radio poses an issue to our model.

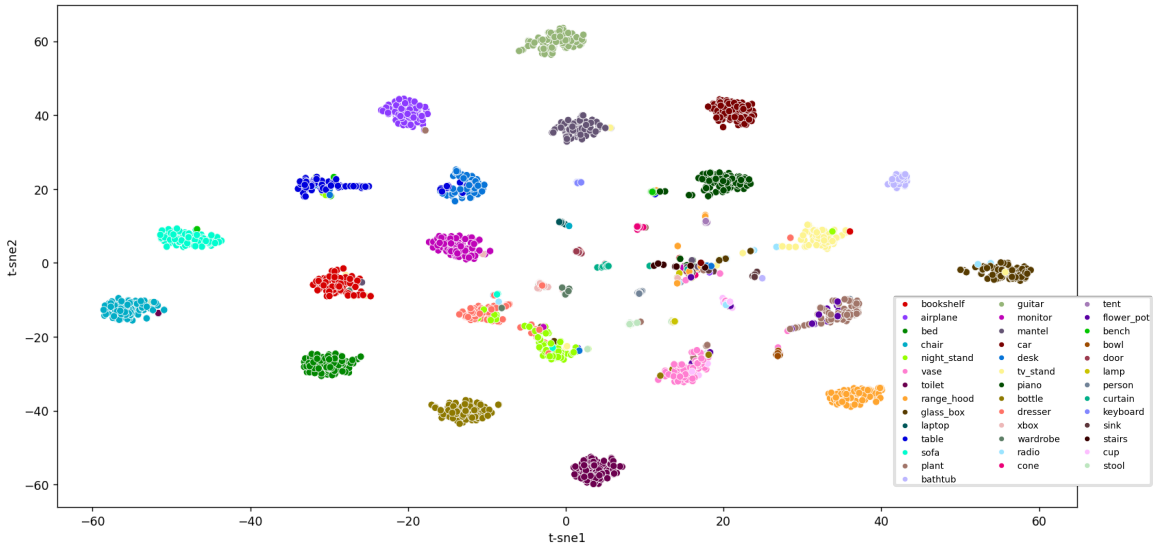


Figure 4.2: t-SNE distribution of model accuracies.

From Figure 4.2 we can see how some classes such as bed, airplane and guitar are clearly clustered away from other classes. However, some classes especially those at the center of the plot are not clustered very well such as night_stand and dresser which have very similar shapes for a lot of their instances.

4.4 Ablation Studies

4.4.1 Disentangled Representation Method

The proposed model orders the points based on two different methods GST and CE, see Section 3.2 for details. In this ablation study, we look at the accuracies and average inference times of these methods across all point clouds. (We used an α-value of 0.04, as it reflects the number of edges that visually makes the graph represent the underlying shape, see Section 4.4.4 for different α-values).

Method	Time (s)	OA	mAcc
GST	0.887	88.15	83.60
CE	0.110	88.31	84.25

Table 4.3: Time (s) is the average time taken in seconds for the sorting of the points using the disentangled representation methods and classification accuracies for GST and CE for geometric disentangled representation on ModelNet40 [1].

From Table 4.3 we can see that CE disentangled representation method is approximately 8 times faster than GST method. Moreover CE method performs slightly better than GST method for both overall accuracy and mean accuracy, thus we choose to move forward with CE method for the following ablation studies.

4.4.2 Neighbor fallback

Since alpha complex does not create a fix number of neighbors for each point/node (see Figure 4.3 for the number of edges per node), there is a need to add a padding

to the input features. We used the following approaches to add padding to the input features:

- **self-loop**: In this method we create self loop with the central node. If we consider a point cloud with N number of nodes then for each node $v_i, i \in N$ we create edge (v_i, v_i) for each missing neighbor node required to get to the selected number of neighbors.
- **knn**: In this method we create an edge between the central node and k nearest neighbor, where k is equal to the number of missing node required to get to the selected number of neighbors.

Padding	**Time (s)**	**Single**		**Dual**	
		OA	**mAcc**	**OA**	**mAcc**
self loop	0.407	88.23	83.56	88.31	84.25
knn	0.778	89.49	85.61	90.95	87.71

Table 4.4: Classification accuracies achieved by using the different padding methods on ModelNet40 [1]. Single refers to the model presented in Section 3.3.1 and Dual represents the model presented in Section 3.3.2. Time (s) refers to the average time taken in seconds to build the graphs for each sample.

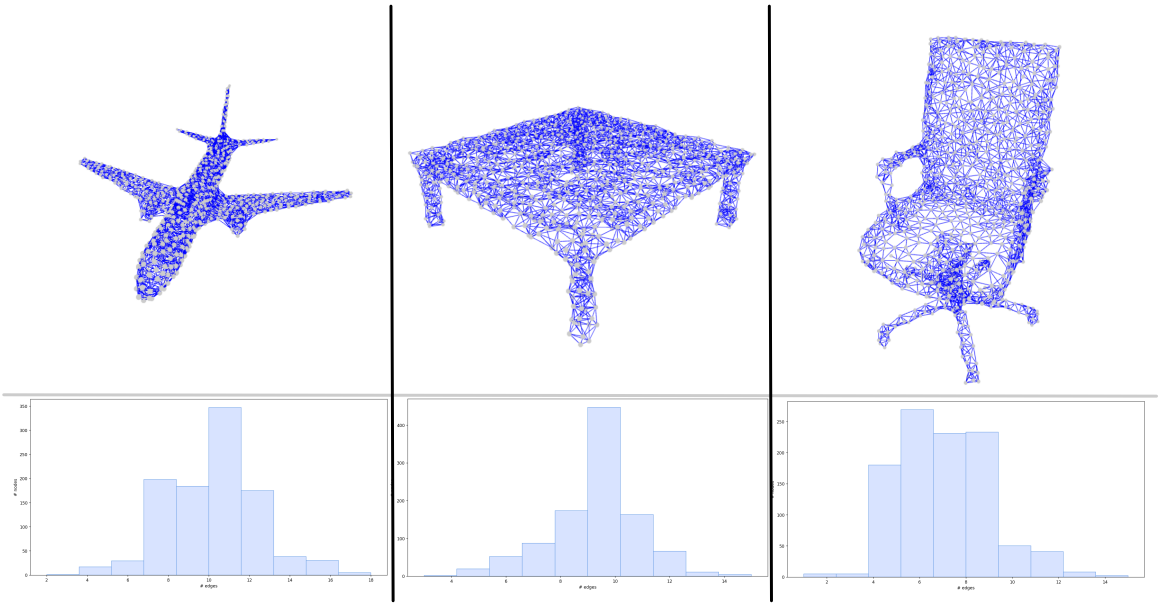


Figure 4.3: Number of edges for each node. α-value=0.04.

As we can see from Table 4.4 that knn padding increases the amount of time required to build the graph. However, the use of knn padding improves the accuracy of our model by a considerable amount. Thus, we will use knn padding as the padding method in the rest of the ablation studies.

4.4.3 Number of Neighbors

Since our model is based on DGCNN [22] which uses tensors that need to be of a fix size in order to learn the corresponding weights and biases, we needed to decide on a fixed number of neighbors for each node. Table 4.6 shows an ablation study on the different number of neighbors. As we can see in Table 4.6 with 16 nodes using the dual-stream model presented in Section 3.3.2, we were able to achieve the highest overall accuracy.

#NN	Single		Dual	
	OA	**mAcc**	**OA**	**mAcc**
10	90.42	85.52	90.75	87.15
16	90.38	86.98	91.03	87.68
20	90.26	86.36	90.95	87.71

Table 4.5: Classification accuracies of using 10, 16 and 20 neighbors on ModelNet40 [1]. Single refers to the model presented in Section 3.3.1 and Dual represents the model presented in Section 3.3.2.

4.4.4 α-value

In this ablation study we experiment with different α-values to see their effects on the model.

α-value	Single		Dual	
	OA	**mAcc**	**OA**	**mAcc**
0.04	88.56	84.30	91.56	88.14
0.05	90.38	86.98	91.03	87.68
2.0 (Delaunay)	89.04	84.22	89.08	85.07

Table 4.6: Classification accuracies of using α-value 0.04, 0.05 and 2.0 (Delaunay) on ModelNet40 [1]. Single refers to the model presented in Section 3.3.1 and Dual represents the model presented in Section 3.3.2.

From Table 4.6, we can see that our dual-stream model performs the best with an α-value of 0.04.

4.4.5 Graph Construction

Based on the previous ablation studies, we can see that using knn as our padding method improves the performance of our model (see section 4.4.2). So in this ablation

study we look at the performance of using knn alone as well as Dynamic knn[1] with graph convolution layers (DGCNN) and GATv2 layers [48] (DGGAN).

Graph	Single		Dual	
	OA	**mAcc**	**OA**	**mAcc**
alpha + knn	88.56	84.30	91.56	88.14
knn	90.63	86.83	90.79	87.19
DGGAN[2]	90.95	88.19	90.22	86.76
DGCNN[3]	91.48	88.57	90.63	87.58

Table 4.7: Classification accuracies achieved by using the different graph construction method on ModelNet40 [1]. The row alpha + knn refers to alpha graph construction with knn padding (see section 4.4.2). knn refers to the use of knn graph construction alone. Single refers to the model presented in Section 3.3.1 and Dual represents the model presented in Section 3.3.2.

Table 4.7 indicates that splitting the input into sharp and gentle region did not improve the accuracy when using dynamic graph construction. This could be due to the latent spaces within the subsequent layers not having any intuitive shape that would benefit from being learned separately. In case of non dynamic graphs, geometric disentangled representation was able to split the shape into different intuitive regions which helped the model to achieve higher accuracies.

4.4.6 Inference Time

In this ablation study we compare the average inference time of our dual-stream model with the different graph constructions and that of other models.

[1]Dynamic knn as presented in DGCNN [22], it is the construction of the knn graph after each layer instead of relying on the previous layer's knn graph structure.

[2]DGCNN [22] model with GATv2 layers [48] instead of the original graph convolution layers.

[3]Accuracy for single stream model is obtained from running the TensorFlow implementation of DGCNN from their GitHub page [50].

Methods	Time (ms)	OA
PointNet [34]	16.6	89.2
PointNet++ [35]	163.2	90.7
DGCNN [22]	27.2	92.9
Ours (alpha)	552.7	88.3
Ours (alpha + knn)	901.9	91.5
Ours (knn)	119.1	90.7
Ours (Dynamic knn)	138	90.6

Table 4.8: Time (ms) is the average inference time in milliseconds taken by each model per sample.

Note the average inference time presented for the other methods in Table 4.8 were taken from DGCNN [22] and they used a different GPU from us. They used two NVIDIA TITAN X GPUs while in our case we used a single NVIDIA Tesla V100-SXM2 GPU.

From Table 4.8, we can see that our alpha complex graph based models (alpha and alpha + knn) took drastically more time than the rest. However, we note that most of the time in all the models were spent carrying out the geometric disentangled representation and graph construction. Since we used Corner and Edge as our geometric disentangled representation we expect based on Table 4.3 that on average around 110ms of our methods' time were spent in that stage. The graph construction stage took the greatest amount of time see Table 4.4 for the average time taken for this stage in case of alpha and alpha + knn. As for knn and Dynamic knn the graph construction is built into the Dual-stream model itself.

Chapter 5

Conclusion and Future Work

In this study, we used alpha complex representation of the point cloud together with a dual-stream model using geometric disentangled representation for 3D object classification. We analyzed and compared the performance of our proposed model on ModelNet40 [1] benchmark dataset, and showed that our model performed close to DGCNN [22] and better than some other state-of-the-art models. However, the accuracy of our approach comes at a cost due to the heavy computational cost of building an alpha complex representation of the data (see Table 4.4 for the average time for building the alpha complex graph for an object).

5.1 Conclusion

In conclusion, we can see that our method performed better than some models as illustrated in Table 4.1. Although it performed better than some model it did not perform better than DGCNN [22] and also have a higher computational cost. A possible reason why our method's accuracy is lower could be the reduced number of edges. Although our initial intuition was that a more coarse and simplified graph representation would provide a better representation of the 3D object, it turns out that having greater number of nearest neighbors is generally better as can be seen from Table 4.4 which implies that using knn was better than simply using self loop

and from Table 4.6 where the accuracy generally went up as the number of nearest neighbor increased. Moreover, the computational cost of building alpha complex representation might be an issue when implemented in real-life applications such as autonomous driving.

5.2 Future Work

In this work, we did not explore the full potential of alpha complex graph. Since alpha complex is based on Delaunay graph we can extract additional information such as the area of the faces and volume of the tetrahedra that remain after alpha complex filtration. Moreover, we used a fixed α-value for all objects while some object might require a different or even varying α-value to get a more accurate graph representation of the object. We could also explore the use of this method in other 3D computer vision tasks such as segmentation and 3D object detection as well as testing the performance of our model on other datasets such as ScanObjectNN [11] and ModelNet40-C [26]. Additionally, we could explore more optimised alpha complex construction methods to address the high inference time of our model.

Bibliography

[1] Wu Z, Song S, Khosla A, et al. 3D ShapeNets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 2015; pages 1912–1920. doi:10.1109/CVPR.2015.7298801.

[2] qizekun. PointVisualizaiton, (url: https://github.com/qizekun/PointVisualizaiton) 2019.

[3] Bello SA, Yu S, Wang C, et al. Review: Deep Learning on 3D Point Clouds. *Remote Sensing* 2020;12(11). ISSN 2072-4292. doi:10.3390/rs12111729.

[4] Xu M, Zhang J, Zhou Z, et al. Learning geometry-disentangled representation for complementary understanding of 3d object point cloud. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35 2021; pages 3056–3064.

[5] Riegler G, Osman Ulusoy A, and Geiger A. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* 2017; pages 3577–3586.

[6] Zeid KA, Schult J, Hermans A, et al. Point2Vec for self-supervised representation learning on point clouds. In *DAGM German Conference on Pattern Recognition*. Springer 2023; pages 131–146.

[7] Ahmed SM, Tan YZ, Chew CM, et al. Edge and corner detection for unorganized 3d point clouds with application to robotic welding. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE 2018; pages 7350–7355.

[8] Tanveer MS, Khan MUK, and Kyung CM. Fine-tuning darts for image classification. In *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE 2021; pages 4789–4796.

[9] Taylor L, King A, and Harper N. Robust and accelerated single-spike spiking

neural network training with applicability to challenging temporal tasks. *arXiv preprint arXiv:2205.15286* 2022;.

[10] Fein-Ashley J, Ye T, Wickramasinghe S, et al. A Single Graph Convolution Is All You Need: Efficient Grayscale Image Classification. *arXiv preprint arXiv:2402.00564* 2024;.

[11] Uy MA, Pham QH, Hua BS, et al. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *Proceedings of the IEEE/CVF international conference on computer vision* 2019; pages 1588–1597.

[12] Bengio Y, Courville A, and Vincent P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 2013;35(8):1798–1828.

[13] Xiao T, Hong J, and Ma J. Elegant: Exchanging latent encodings with gan for transferring multiple face attributes. In *Proceedings of the European conference on computer vision (ECCV)* 2018; pages 168–184.

[14] Kipf TN and Welling M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* 2016;.

[15] Veličković P, Cucurull G, Casanova A, et al. Graph attention networks. *arXiv preprint arXiv:1710.10903* 2017;.

[16] Izadi MR, Fang Y, Stevenson R, et al. Optimization of graph neural networks with natural gradient descent. In *2020 IEEE international conference on big data (big data)*. IEEE 2020; pages 171–179.

[17] Zhang M, Cui Z, Neumann M, et al. An end-to-end deep learning architecture for graph classification. AAAI'18/IAAI'18/EAAI'18. AAAI Press. ISBN 978-1-57735-800-8 2018; .

[18] Ma Y, Wang S, Aggarwal CC, et al. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* 2019; pages 723–731.

[19] Gao H and Ji S. Graph representation learning via hard and channel-wise attention networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* 2019; pages 741–749.

[20] Wu Z, Pan S, Chen F, et al. A Comprehensive Survey on Graph Neural Networks.

IEEE Transactions on Neural Networks and Learning Systems 2021;32(1):4–24. ISSN 2162-2388. doi:10.1109/tnnls.2020.2978386.

[21] Ye X, Li J, Huang H, et al. 3D Recurrent Neural Networks with Context Fusion for Point Cloud Semantic Segmentation. In Ferrari V, Hebert M, Sminchisescu C, et al., editors, *Computer Vision – ECCV 2018*. Springer International Publishing, Cham. ISBN 978-3-030-01234-2 2018; pages 415–430.

[22] Wang Y, Sun Y, Liu Z, et al. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)* 2019;38(5):1–12.

[23] Srivastava S and Sharma G. Exploiting Local Geometry for Feature and Graph Construction for Better 3D Point Cloud Processing with Graph Neural Networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* 2021; pages 12903–12909. doi:10.1109/ICRA48506.2021.9561327.

[24] Singh P, Sadekar K, and Raman S. TreeGCN-ED: encoding point cloud using a tree-structured graph network. *arXiv preprint arXiv:2110.03170* 2021;.

[25] Gomes P, Rossi S, and Toni L. Spatio-temporal graph-RNN for point cloud prediction. In *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE 2021; pages 3428–3432.

[26] Sun J, Zhang Q, Kailkhura B, et al. Benchmarking robustness of 3d point cloud recognition against common corruptions. *arXiv preprint arXiv:2201.12296* 2022; .

[27] De Deuge M, Quadros A, Hung C, et al. Unsupervised feature learning for classification of outdoor 3D scans 2013;2(1).

[28] Dai A, Chang AX, Savva M, et al. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition* 2017; pages 5828–5839.

[29] Su H, Maji S, Kalogerakis E, et al. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision* 2015; pages 945–953.

[30] Qi CR, Su H, Nießner M, et al. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition* 2016; pages 5648–5656.

[31] Kanezaki A, Matsushita Y, and Nishida Y. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In

Proceedings of the IEEE conference on computer vision and pattern recognition 2018; pages 5010–5019.

[32] Maturana D and Scherer S. VoxNet: A 3D Convolutional Neural Network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* 2015; pages 922–928. doi:10.1109/IROS.2015.7353481.

[33] Le T and Duan Y. PointGrid: A Deep Network for 3D Shape Understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 2018; .

[34] Charles RQ, Su H, Kaichun M, et al. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 2017; pages 77–85. doi:10.1109/CVPR.2017.16.

[35] Qi CR, Yi L, Su H, et al. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems* 2017;30.

[36] Zhao H, Jiang L, Fu CW, et al. PointWeb: Enhancing Local Neighborhood Features for Point Cloud Processing. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 2019; pages 5560–5568. doi:10.1109/CVPR.2019.00571.

[37] Yang J, Zhang Q, Ni B, et al. Modeling Point Clouds With Self-Attention and Gumbel Subset Sampling. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 2019; pages 3318–3327. doi:10.1109/CVPR.2019.00344.

[38] Klokov R and Lempitsky V. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proceedings of the IEEE international conference on computer vision* 2017; pages 863–872.

[39] Wang W, You Y, Liu W, et al. Point cloud classification with deep normalized Reeb graph convolution. *Image and Vision Computing* 2021;106:104092. ISSN 0262-8856. doi:https://doi.org/10.1016/j.imavis.2020.104092.

[40] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations* 2021; .

[41] Yu X, Tang L, Rao Y, et al. Point-BERT: Pre-training 3D Point Cloud Transformers with Masked Point Modeling. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 2022; pages 19291–19300. doi: 10.1109/CVPR52688.2022.01871.

[42] Devlin J, Chang MW, Lee K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein J, Doran C, and Solorio T, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota 2019; pages 4171–4186. doi:10.18653/v1/N19-1423.

[43] Baevski A, Hsu WN, Xu Q, et al. Data2vec: A general framework for self-supervised learning in speech, vision and language. In *International Conference on Machine Learning*. PMLR 2022; pages 1298–1312.

[44] He K, Chen X, Xie S, et al. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* 2022; pages 16000–16009.

[45] Maur P. Delaunay Triangulation in 3 D 2010; .

[46] Chen D, Li J, Di S, et al. Critical Points Extraction from Building Façades by Analyzing Gradient Structure Tensor. *Remote Sensing* 2021;13(16). ISSN 2072-4292. doi:10.3390/rs13163146.

[47] Mishra S, Sarkar U, Taraphder S, et al. Principal Component Analysis. *International Journal of Livestock Research* 2017;page 1. doi:10.5455/ijlr.20170415115235.

[48] Brody S, Alon U, and Yahav E. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491* 2021;.

[49] Kingma DP and Ba J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* 2014;.

[50] WangYueFt. Dynamic Graph CNN for Learning on Point Clouds (url: https://github.com/WangYueFt/dgcnn) 2019.

[51] Harris CR, Millman KJ, van der Walt SJ, et al. Array programming with NumPy. *Nature* 2020;585(7825):357–362. doi:10.1038/s41586-020-2649-2.

[52] Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 2011;12(Oct):2825–2830.

[53] Virtanen P, Gommers R, Oliphant TE, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 2020;17:261–272. doi:10.1038/s41592-019-0686-2.

[54] Abadi M, Agarwal A, Barham P, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems 2015. Software available from tensorflow.org.

[55] Zhou QY, Park J, and Koltun V. Open3D: A Modern Library for 3D Data Processing 2018.

[56] Hunter JD. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 2007;9(3):90–95. doi:10.1109/MCSE.2007.55.