Multi Abstraction Geodesic Neighbourhood-based network(magNet) for point clouds understanding

A Project Report

submitted by

AJINKYA GANESHRAO AMBATWAR

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

JUNE 2021

THESIS CERTIFICATE

This is to certify that the thesis titled **Multi Abstraction Geodesic Neighbourhood-based network(magNet) for point clouds understanding**, submitted by **Ajinkya Ganeshrao Ambatwar**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master Of Technology(M.Tech)**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Anurag Mittal
Research Guide
Associate Professor
Dept. Of Computer Science and Engineering
IIT-Madras, 600 036

Prof. K.Mitra
Research Co-Guide
Associate Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 29/06/2021

ACKNOWLEDGEMENTS

Firstly, I would like to sincerely thank my advisor Dr. Anurag Mittal. He has given me the opportunity to pursue my interest in Computer Vision. He has been patient and supportive through the ups and downs of my journey leading up to this thesis. With his able guidance, my advisor has helped me get through research predicaments on multiple occasions. Apart from research, I have also learnt valuable lessons on how to thrive in an academic environment and in life in general. I am grateful for having the opportunity to work closely with my advisor.

I am also thankful for having the opportunity to interact with the professors of Electrical Enginering and computer science department through coursework and otherwise. I am also thankful for the interactions that continued after the completion of the course which made my research journey easier. I am also thankful to Dr. Kaushik Mitra, my co-guide for this project, who have been supportive in completion of my thesis. My research journey has helped me forge new bonds and strengthen the older ones. I am grateful for my labmates whose support and kinship has made my time with them memorable. I would like to offer my sincere gratitude to current and previous students of CV lab with specific mentions of Nagender Vasireddy and Arulkumar S. who have been extremely supportive in my journey to the completion of this project.

I would like to offer my thanks to all the teaching and non-teaching staff of IIT Madras for providing me the best possible environment to grow during my 5 years at IIT Madras.

A special mention of all the amazing friends I got during my 5 years at IIT Madras is must. They have been totally supportive and encouraging during all ups and downs in my life.

Last but not least, I am grateful to be part of my family. Despite their own difficulties, they have always been supportive of my pursuit of interests and growing up along the way. I cannot be grateful enough for their immense patience and forgiving nature without which my life would not be the same.

ABSTRACT

KEYWORDS: point cloud; geodesic neighbors; 3D object detection; part segmentation

Convolutional neural networks (CNNs) have significantly improved the performance of computer vision tasks with 2D images as input. Similar boost in performance has not extended to point clouds of objects as the points are not spatially ordered and are variable in number. In our work, we present CNN model with Geodesic Neighbourhood based feature descriptors abstracted at multiple scales for better understanding of 3D input. The input to the model is a 3D point cloud with an irregular structure. In the multilayered network, in each layer, the points are sampled and then geodesic neighbourhood is calculated for each point in the sampled space. The point feature of each point is generated based on the calculated geodesic-neighbourhood of the point. A geodesic distance-based local neighbourhood is determined using distance along paths that lie on the surface of the object. Such a local neighbourhood makes the point feature descriptor invariant to shape articulation about some joint locations.

In this work, we introduce a hierarchical neural network that applies a 2D concolution based network recursively on a nested partitioning of the input point set. The nested partition is calculated using effective geodesic neighbourhood approximation in the 3D graph models of the object. We develop a custom **PyTorch Extension** using cuda kernels that we build from scratch that does the computation of geodesic neighbouhood for each point during forward path of the model. This library can act as an effective base for further future works on this and similar domain.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
LIDAR	Light Detection And Ranging
Radar	Radio detection and ranging
CNN	Convolutional Neural Network
RoI	Region of Interest
FoV	Field of View
CAD	Computer-aided Design
magNet	Multi-Abstraction Geodesic-neighbouhood based network
FPS	Farthest Point Sampling
AM	Abstraction Module
FP	Feature Propagation Module

NOTATION

r **or** T_g	Threshold for Geodesic distance
T_e	Threshold for Eulidean distance in PointNet++
P_{pg}	The set of Geodesic neighbourhood of a point
N'	Number of output points after Sampling Layer
K	Maximum Number of geodesic neighbours for a point
R^3	Euclidean Space of Dimension 3
R^3_d	Geodesic metric Space
P_{us}	Unsampled Point Set
P_s	Sampled Point Set
f_{us}	Unsampled Point Set Features
f_s	Sampled Point Set Features
Id_{geo}	Indices of the geodesic neighbourhood of a point
f_{geo_s}	Features of the geodesic neighbourhood of a Sampled point set
f_{ps}	Feature of entire point cloud

CHAPTER 1

INTRODUCTION

Processing 3D point cloud data accurately is crucial in many applications including autonomous driving(Navarro-Serment *et al.* (2010)) and robotics(Bogdan Rusu *et al.* (2009)). In these settings, sensors like LIDAR produce unordered sets of points that correspond to object surfaces. Correctly classifying objects from this data is important for 3D scene understanding(Uy *et al.* (2019)). To understand a 3D scene means to identify the objects present in the scene, to local- ize the identified objects in 3D space, to understand the scene layout and to understand the interplay between different elements of the scene. Thus, 3D scene understanding can be further divided into subtopics such as 3D object detection, 3D scene semantic segmentation, ego-motion estimation, 3D reconstruction etc. In this thesis, we review the deep learning based methods for object classification and part segmentation of 3D point clouds and suggest a new method which addresses the drawbacks in existing methods.
While classical approaches for this problem have relied on hand-crafted features (Arras *et al.* (2007)), recent efforts have focused on the design of deep neural networks (DNNs) to learn features directly from raw point cloud data (Qi *et al.* (2017*a*)). Deep Learning-based methods have proven effective in aggregating information across a set of 3D points to accurately classify objects.

The most widely adopted benchmark for comparing methods for point cloud classification has been ModelNet40. The accuracy on Mod elNet40 on the classification tas has steadily improved over the last few years from 89.2% by PointNet (Qi *et al.* (2017*a*)) to 94.2% by CurveNet (Xiang *et al.* (2021))[Fig. 1.1].

1.1 Object classification and part segmentation on point clouds

Given a 3D scene, 3D object detection is the task of locating an object in the scene and identifying the type of the object. Thus, 3D object detection can be further classi-

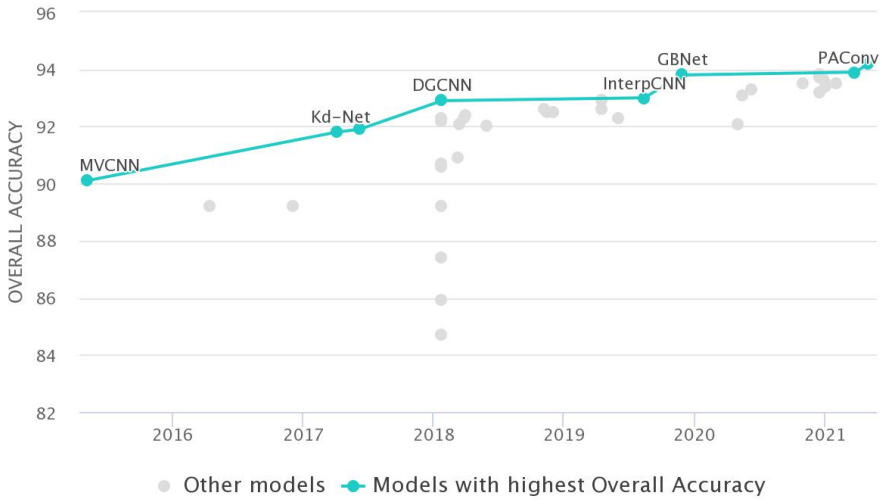


Figure 1.1: Performance on ModelNet40 classification task over time.

fied into subtasks such as 3D object proposals and 3D object classification. 3D object proposal methods suggest possible locations of objects in a point cloud of a 3D scene whereas 3D object classification methods identify the class of an object. Input to a method that performs 3D object classification using point cloud data is the isolated point cloud of an object and the output is the predicted label of a class to which the object belongs.

Part segmentation is the task of identifying the regions occupied by parts of the object. In 2D images, this task is carried out by classifying each of the pixel in the region of the object with a set of labels corresponding to the parts of the objects. Similarly, 3D part segmentation using point clouds of objects is carried out by classifying each point in the point cloud to the corresponding part of the object. In the case of 3D part segmentation, the input to the task is a point cloud of the object and output is the predicted part label of each point in the point cloud.

1.2 Challenges of applying deep learning methods on 3D point clouds

The boost in performance seen in computer vision tasks with images since the introduction of deep learning methods has not extended to computer vision tasks using point clouds. Deep learning methods are less effective with point clouds as input which can

be attributed to the structure of point cloud data and non-availability of large scale 3D point cloud datasets. A point cloud is a set of points of variable cardinality as opposed to an image which is structured into a 2D grid of pixels. The 2D grid structure of pixels in images make it viable to apply the 2D spatial convolution operation on them. On the other hand, each point cloud in a real-world dataset is represented as a sequence of unordered point coordinates with variable number of points in each point cloud. The spatial distribution of the points in a point cloud is nonuniform due to varying spatial density within the point cloud. The variable lengths of the sequences of points combined with their varying spatial densities make it unsuitable for straightforward application of spatial convolution operations on point cloud data. Since a convolution operation is one of the core aspects of deep learning methods in computer vision tasks, lack of availability of this operation for point cloud data is one of the major drawbacks which hindered the effectiveness of deep learning methods on point cloud data.

1.3 Related Works

The initial work in this domain was VoxNet(Maturana and Scherer (2015)). VoxNet, comprises of 3D CNN layers followed by fully connected layers which gives a softmax output to classify the input object into its respective class.

Qi *et al.* (2017*a*) introduced the seminal PointNet architecture for the task of object classification and part segmentation with point cloud data as input. This architecture outperforms VoxNet architecture in the object classification task on ModelNet40 dataset as it utilises all of the point locations within the point cloud, thus utilising the fine shape information discarded by VoxNet. It uses pointwise convolutional layers to process the point sets independently of each other. Each point is represented by a feature vector based on its spatial location. The output of final 1D convolutional layer is fed to a *symmetric function*. The symmetric function used in their work is *ElementWise MaxPool Operator*.

Architectures focusing on Point Cloud Understanding can be categorically separated in 3 types a) voxel based b) Point Based c) Graph Based.

An approach to transforming irregular point clouds to regular representations is 3D voxelization, followed by 3D convolution in 3D. When applied naively it has been ob-

served that these methods incur massive computation and memory costs due to the cubic growth in the number of voxels as a function of resolution. The sparsity of voxels causes memory inefficiency and processing overhead. Few prior works use this sparsity for efficient processing. OctNet(Riegler *et al.* (2017)) uses unbalanced octree with hierarchical partitions. Approaches based on sparse convolutions, where the convolution kernel is only evaluated at occupied voxels, can further reduce computation and memory requirements.(Graham *et al.* (2018); Kanezaki *et al.* (2018)).

A broad class of DNNs have emerged to process 3D point clouds directly (Simonovsky and Komodakis (2017); Zaheer *et al.* (2017); Klokov and Lempitsky (2017); Xu *et al.* (2018); Atzmon *et al.* (2018); Li *et al.* (2018); Ben-Shabat *et al.* (2018); Qi *et al.* (2017*b*)). PointNet proposed one of the first strategies, where features are updated for each point with MLP layers, and aggregated with global max pooling. However, no local comparisons are performed in PointNet,which motivated PointNet++(Qi *et al.* (2017*b*)). PointNet++ uses a hierarchical neural network that applies PointNet recursively on a nested partitioning of the input point set. RSCNN(Liu *et al.* (2019)) uses MLPs conditioned on the spatial relationship of two points to update and aggregate features around an individual sampled point. There exist many variations to these methods, but the emerging trend is an increase in network sophistication.

Graph based approaches for 3D point cloud classification and object detection have been proposed in recent works(Chen *et al.* (2020); Lin *et al.* (2020); Shi *et al.* (2020)).Chen et al. uses shape attentive U-shape Graph Convolution network to address the variable scale of objects in the point cloud(mainly for object detection). Lin et al. proposes a deformable kernel methods that directly work on local graphs created inside the point cloud. Shi et al. assumes the point cloud as a graph with vertex and edges. It uses multiple iteration of a graph neural network layer which updates the edge feature and vertex feature using autoregistration mechanism and MLP layers.

The main downfall of many of these works is that they don't consider local point interaction between the points. Hence in our work we propose a local feature descriptor based on pointNet. We represent each point with a feature constructed from the relative locations of the set of points within its local neighbourhood. We use a Geodesic Distance based neighbourhood approach for local feature embedding. Like PointNet++, the geodesic neighbourhood is calculated on a partitioned subset of the point cloud.

1.4 Proposed solution to use all the points and local shape feature representation

In this thesis, we present a local shape feature descriptor based neural network which depends on the geodesic neighbourhood of the point set. The proposed network addresses the following drawbacks

- VoxNet - Underutilisation of all points in the point cloud
- PointNet - Local Point interaction not captured
- 3D CNN based models - Overutilization of computation resources due to sparsity in point cloud
- PointNet++ - Previous studies by Nagender et al. 2019(IIT Madras), showed that the point feature generated using Euclidean distance-based local neighbourhood points is not invariant to rotations of the object and articulations of the object near joint locations. The Euclidean neighbourhood of a point might contain points that correspond to parts of the object that are disconnected within the local neighbourhood. Hence we use a geodesic neighbouhood based idea.

The input to each layer in the network is the point cloud, 3D model surface data and the model graph properties of the 3D object. The surface data and the graph properties of the corresponding point cloud are needed to calculate the geodesic neighbourhood of the point set. Geodesic distance between two points in a point cloud is the distance between the points that is measured along a path that lies on the surface of the object. In each layer, the point set is first sampled into a subset. The geodesic neighbourhood set calculated for each sampled point is passed through a set of 2D CNN layers. Since the input to this layer are the set of relative 3D coordinates of the local geodesic neighbourhood points, the output is a shape feature representing the shape formed by the neighbourhood points.

The sampling of point clouds provides us an implicit multi abstraction. The initial complete point cloud will provide more detailed information about the shapes in a smaller locality. As we go on sampling the point cloud, the RoI will enlarge and information about an overall larger scale is obtained.

The reason behind using geodesic distance instead of standard euclidean distance as in PointNet++ is following:

The point feature generated with a Euclidean distance-based local neighbourhood is translation-invariant since the relative coordinates of the local neighbourhood points are invariant to object translations. However, the Euclidean local neighbourhood of a point might contain points that belong to adjacent disconnected parts of the object. Relative motion between these adjacent disconnected parts of the object could result in inconsistent set of points in the local neighbourhood. A geodesic distance-based local neighbourhood is articulation-invariant around joint locations in an object in addition to being translation-invariant.

We have discussed so far about introducing a novel point based network for the task of point cloud understanding which takes into account local shape features found using geodesic neighbourhood. In the following chapter, we shall discuss about the Point Based methods for 3D point cloud understanding. This will provide us intuition behind the proposed method and help us build upto it.

CHAPTER 2

REVIEW OF POINT BASED METHODS FOR 3D POINT CLOUD UNDERSTANDING

In this chapter, we review some of the existing deep learning based methods for point cloud classification and part segmentation tasks with point cloud of an object as input. We shall specifically focus on the point based methods for this task. Although deep learning methods are a recent addition to the point cloud literature, several approaches have been proposed to make training a deep learning model viable on point cloud datasets.

2.1 Point Based Methods

In this section, we describe the methods that use the points in a point cloud as input to a deep learning model without transforming them into other representations of the object. We shall particularly focus on two architectures viz. PointNet and PointNet++. One of the advantages of such methods is the prevention of loss of information during the process of transformation of the point cloud into other representations of the object. The first of such point-based methods, PointNet, was introduced by Qi *et al.* (2017*a*). In the PointNet architecture, the dimension of the vector representing each point is increased using a sequence of 1D convolution layers. The resulting vectors corresponding to each point in the point cloud are used as input to a symmetric function to generate the point cloud feature.The architecture of PointNet is briefly described in Section 2.2. The main contribution of Qi *et al.* (2017*a*) is a deep learning model which can take a variable number of unordered points in a point cloud as input and generate a point cloud feature which captures the shape of the corresponding object. The PointNet architecture showed significant improvement in performance in computer vision tasks over the then state-of-the-art deep learning based models with point clouds of objects as input.

One of the major drawbacks of PointNet is the lack of interaction between the points and its neighbours before being used as input to the symmetric function. The dimen-

sionality of the vector corresponding to each point in the point cloud is increased independently as described in Section 2.2. Several methods have been proposed to address this issue by using PointNet-like architectures hierarchically in local neighbourhoods of points in each layer. We shall discuss the PointNet++ Qi *et al.* (2017*b*) in Section 2.3 which fundamentally uses the local euclidean neighbourhood for each point.

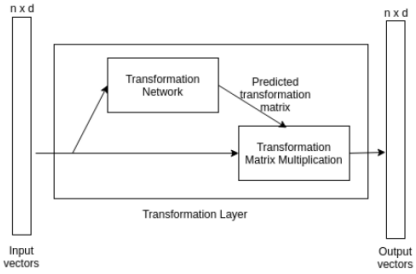
2.2 PointNet

In this section, we describe the seminal PointNet architecture introduced by Qi *et al.* (2017*a*)The input to this architecture is the point cloud of an object and the output is a point cloud feature representing the input point cloud. The architecture consists of 1D convolution layers which increase the dimensionality of the vector representing each point, feature transform layers which introduce invariance in the network to object translation and rotation, and a symmetric function which generates the point cloud feature from the higher dimensional representations of each point. In the PointNet architecture, the symmetric function is an element-wise maxpool layer.The inputs to the element-wise maxpool layer are the vectors representing each of the points in the higher dimensional space and the output is a feature in the higher dimensional space. The output feature of the element-wise maxpool layer is generated according to the Equation 2.1, where $out \in R^d$ is the output of the element-wise maxpool layer, $inp \in n \times R^d$ is the input to the element-wise maxpool layer, n is the number of points in the input and d is the dimension of all of the vectors representing each point and the output feature.

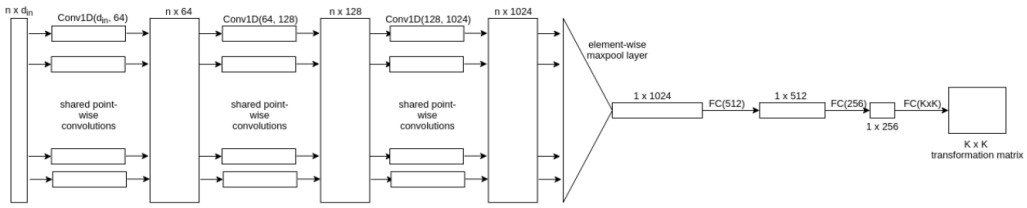
We denote each feature transform layer as $FT(K)$, where K × K is the dimension of the transformation matrix applied to each vector of dimension K representing a point in the input to the transform layer.

$$out[j] = \max_{1 \leq i \leq n} inp[i, j], \qquad \forall j = 1, 2, ...d \tag{2.1}$$

The input to the feature transform layer $FT(K)$ is the set of vectors of dimension K corresponding to each point in the point cloud. The feature transform layer consists of a transformation network which predicts the K × K transformation matrix. The transformation network consists of a PointNet-like architecture with 1D convolution



(a) A representation of the transformation layer



(b) A representation of the transformation network used in the transformation layer

Figure 2.1: A representation of the transformation layer is shown in Figure (a). A representation of the transformation network used to predict the transformation matrix in the transformation layer is shown in Figure (b). FC(n) is a fully connected layer with n neurons.

layers, a symmetric function and fully connected layers as shown in Figure 2.1b A representation of the transformation layer is shown in Figure 2.1a.

In the PointNet architecture, the 3D coordinates of points in the input point cloud are first sent through a feature transform layer, $FT(3)$. The dimension of the vector representing each point is increased to 64 using a sequence of two 1D convolution layers, $Conv$1D(3, 64) and $Conv$1D(64, 64). The 1D convolution layers act independently on each point in the point cloud. The resulting points with 64-dimensional vectors are passed through another feature transform layer, $FT(64)$. The resulting points with 64-dimensional vectors are passed through a sequence of three 1D convolution layers, $Conv$1D(64, 64), $Conv$1D(64, 128) and $Conv$1D(128, 1024). A 1024-dimensional point cloud feature representing the point cloud is generated by passing the output of the final 1D convolution layer through a symmetric function. The representation for the entire PointNet architecture is shown in Figure

One major visible concern with respect to PointNet is lack of local shape structure understanding. The points are processed independent of each other and the symmetric function operates on the point features. Local point interaction is important for better

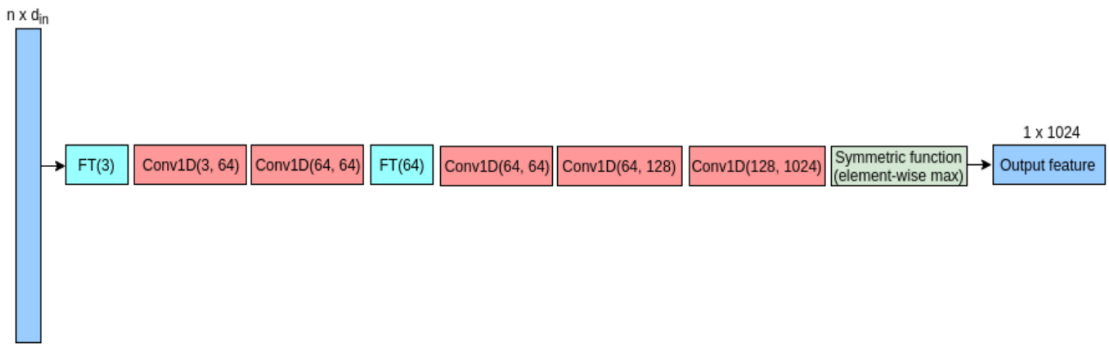


Figure 2.2: A representation of the PointNet architecture.

representation of the point cloud as a whole. PointNet++ tackles thsi problem of local feature representation. This forms the very basis of our proposed architecure as well. We shall have brief discussion about PointNet++ in next section.

2.3 PointNet++

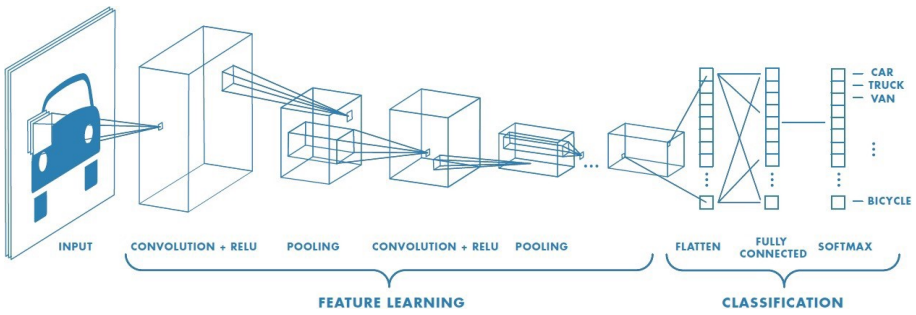
In this section, we shall discuss PointNet++(Qi *et al.* (2017*b*)). This architecture focuses on solving one major drawback of the PointNet architecure which is local point interaction. Exploiting local structure has proven to be important for the success of convolutional architectures. The fundamental idea of PointNet++ is similar to the receptive field of a standard CNN network. In PointNet++, a neural netowork takes in the point data normalized in a sphere as the input and progressively captures features and increasing scale along a multi-resolutions hierarchy. In the initial stages, the network has a smaller FoV and learns a more local set of features while at later stages the FoV is large. Such an ability of the network allows better generalizability to unseen cases.

The point set in first sampled into overlapping local regions by the distance metric of the underlying space. Similar to CNNs, local features capturing fine geometric structures from small neighborhoods are extracted; such local features are further grouped into larger units and processed to produce higher level features. This process is repeated until the features of the whole point set are obtained. Analogy between CNN and PointNet++ can be seen from Figure 2.3

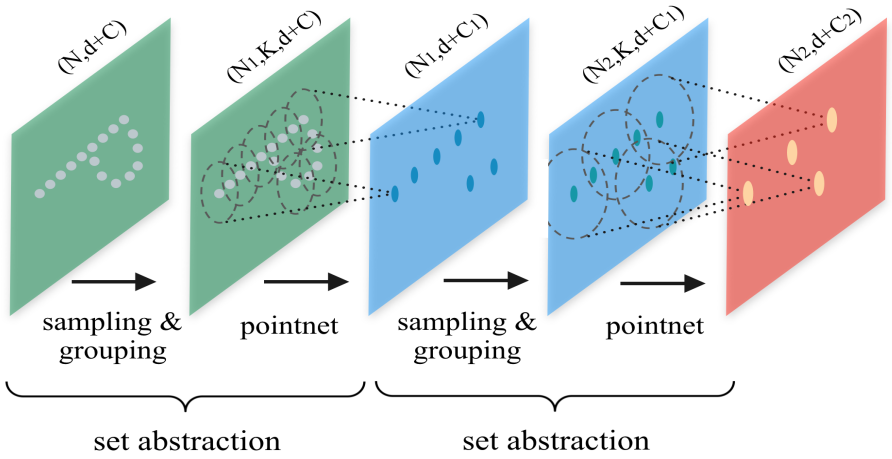
The design of PointNet++ addresses two issues: 1) The problem os efficient partitioning of the point set and 2) The of abstraction of points sets or local features through

a local feature learner. The two issues are correlated because the parititioning should produce similar structures across partitions and learn in a shared weights fashion similar to convolutional networks. The local feature learner is set to PointNet(refer to Section 2.2). As a basic building block, PointNet learns the local feature embeddings and progresses it to higher levels. In this view, PointNet++ applies PointNet recursively on a nested partitioning of the input set.

For generating the overlapping partition of the point set. The partition is created around the centroid location using a euclidean neighbourhood equal to the scale(T_e). PointNet++ uses FPS(Moenning and Dodgson (2003)) to sample the centroid locations. FPS has proven to have a better point coverage than the random uniform sampling.



(a) A standard CNN network



(b) PointNet++ sampling

Figure 2.3: An anlogy between CNN network and PointNet++. Similar to CNN, the PointNet++ looks at the input point set with increasing receptive field

Another problem that PointNet++ addresses is variable point density on the point sets. PointNet++ uses MultiScale Grouping which concatenates features obtained from multiple scales. If in some region the point density is lesser, such a mechanism makes sure there are enough points in the neighbouhood set.

2.3.1 Hierarchical Point Set Feature Learning

Unlike PointNet in which a single symmetric function aggregates the entire point set, PointNet++ architecture builds multi-hierarchical grouping of points with increasing field of view by increasing the scale.

The hierarchical structure in composed by a number of *set abstraction* levels. Each set abstraction layer consists of the *Sampling* layer, *Grouping* layer and *PointNet* layer. The *Sampling* layer samples few points from the input set. The sampled points form the centroids of local regions. *Grouping* layer then constructs the local region sets for the sampled centroids using ***Euclidean*** metric in the unsampled space. *PointNet* layer uses a mini-PointNet to encode local region patterns into feature vectors.

Mathmatically speaking, a set abstraction layer takes an input of dimension $N \times (d + C)$ where N is the number of input points with d-dim coordinates and C-dim point features. The input passes through the ***Sampling*** layer which outputs N' indices. Given input points $\{x_1, x_2, ..., x_N\}$, iterative FPS is applied to choose the subset $\{x_{i_1}, x_{i_2}, ..., x_{i_{N'}}\}$, such that x_{i_j} is the most distant point(in euclidean space) from the set $\{x_{i_1}, x_{i_2}, ..., x_{i_{j-1}}\}$ with regard to rest points. ***Grouping*** layer aggregates the euclidean neighbourhood for each of the point and generates the $N' \times K \times (d + C)$ dimension matrix. Here K represent the number of euclidean neighbours for each point. The ***PointNet*** layer takes this $N' \times K \times (d + C)$ dimension matrix and returns $N' \times (d + C')$ dimension output, where C' is the output feature dimension.

Regarding the variable density in point cloud, PointNet++ uses **Multi-scale Grouping** (Figure. 2.4) Where in the ***Grouping*** layer multiple radii are used to calculate the euclidean neighbourhood and the features are concatenated. This will take into account the lack of sufficient points scenario in case of lesser point density and produce better results.

A full representation PointNet++ is shown in Figure. 2.5

In this chapter, we built the basic understanding of Point based networks for point cloud understanding. We particularly focused on PointNet and PointNet++. We understood that local shape features are of important for better understanding of the point set. Till now we have seen Euclidean distance based neighbourhoods for local shape feature description. In next chapter, we introduce another effective local feature de-

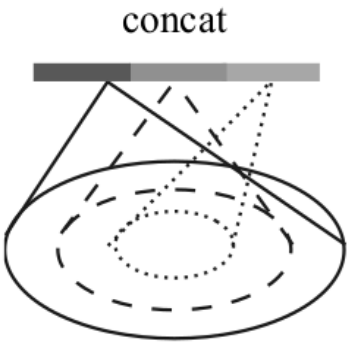


Figure 2.4: MultiScale Grouping

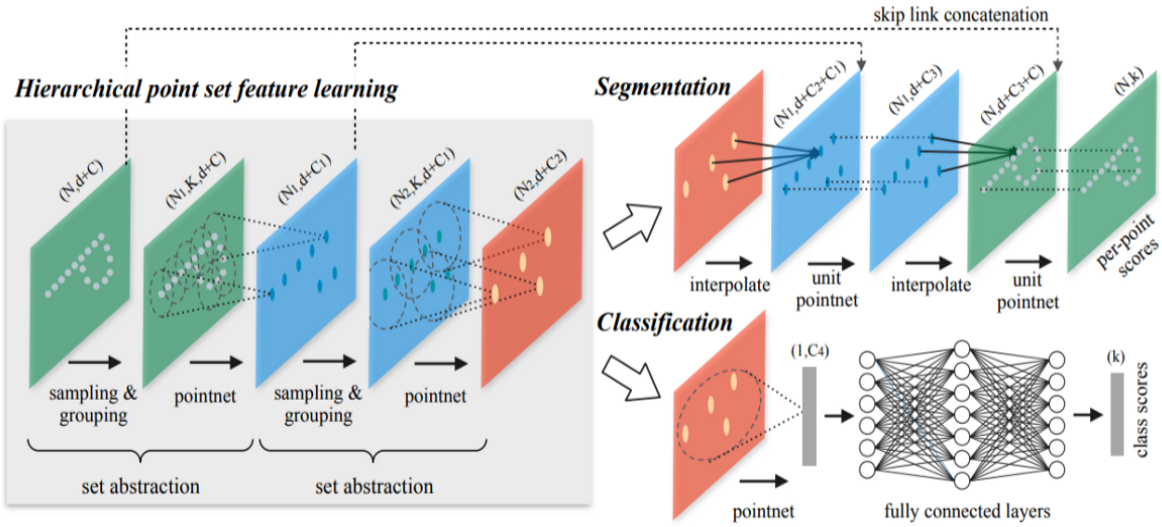


Figure 2.5: PointNet++ with classification and segmentation networks

scriptor based on Geodesic distance neighbourhood. We propose a similar architecture to PointNet++ using this feature descriptor in the next chapter.

CHAPTER 3

Multi Abstraction Geodesic Neighbourhood-based network(magNet)

So far we have dicussed the functioning of Point based networks and got an intuition behind their architecure. We observed how PointNet uses entire point set and processes it using Transformation networks and use an element-wise maxpool layer as a symmetric function to aggregate the point feature. We observed the lack of local point interaction in the PointNet that causes lack of understanding of underlying struture in the point set. We saw how PointNet++ tackles the problem by using Euclidean distance based local shape feature descriptors and hierarchically apply it to get the shape feature at multiple scales.

Recent works by Nagender et.al(2019)(IIT Madras) talk about the issues associated with using Euclidean Neighbourhood for the points. Using that intuition, we propose a novel method for robust and more intuitive understanding of Point Clouds. We call it **Multi Abstraction Geodesic Neighbouhood-based network(magNet)**. Let us first understand the meaning behind nomenclature. **"Multi Abstraction"** refers to the fact that we shall work on multiple levels of abstraction similar to PointNet++. **"Geodesic Neighbouhood"** refers to using Geodesic Distance for finding the local neighbouhood for each point.

The rest of this report goes as follows. We shall first discuss about issues with Euclidean Neighbourhood in Section 3.1. In Section 3.2 we briefly discuss the concept of geodesic distance. In Section 3.3 we discuss our proposed architecure.

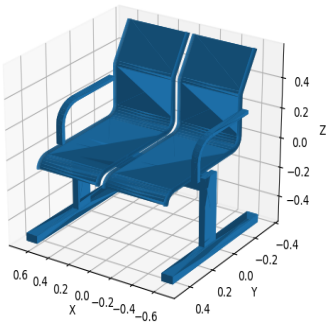
3.1 Issues with Euclidean Neighbouhood

The euclidean local neighbourhood based point feature is translation invariant as it is dependent only on the relative location of the points within the local neighbourhood. The translation invariance property of the point feature increases the learning capacity

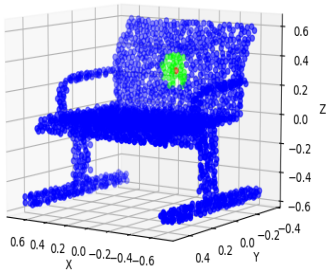
of the network as similar local parts across all of the objects can be represented with similar features which makes training easier.

However, the point feature generated using Euclidean distance-based local neighbourhood points is not invariant to rotations of the object and articulations of the object near joint locations. The Euclidean neighbourhood of a point might contain points that correspond to parts of the object that are disconnected within the local neighbourhood. Examples for the same are shown in Figure. 3.1 and Figure. 3.2. We can observe from the Fig. 3.1, near the point of interest, euclidean neighbourhood consists of points from the other bench(left bench) as well which is not directly connected to it, while Geodesic neighbourhood considers only the points that make the part of the right bench of which the point of interest is a part of. Similar observations can be obtained from Fig. 3.2, where the point from the tyre is under observation. We can observe that, though Euclidean neighbourhood has more number of points but few of the points are included from the front hood region which does make the part of the tyre. But geodesic neighbourhood although having lesser number of points, is strictly confined to the tyre region of the car.

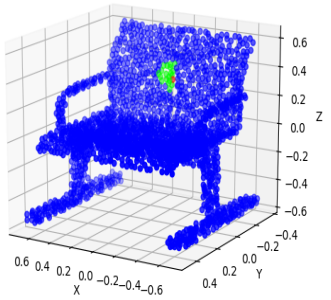
This illustration clearly provides us the intuition behind moving towards geodesic neighbourhood from euclidean neighbourhood. Following up on this, in the next section we shall briefly discuss about the concept of Geodesic distance and Geodesic neighbourhood.



(a) Surface Plot Of Bench

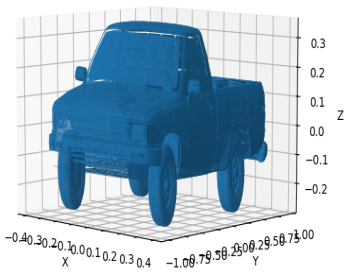


(b) Point cloud (blue) of a bench showing Euclidean neighbourhood (green) of a point (red)

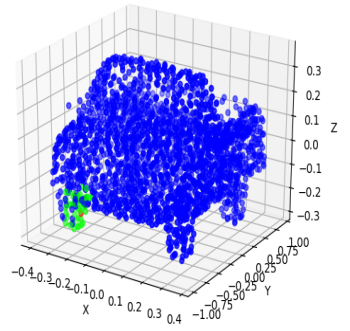


(c) Point cloud (blue) of a bench showing Geodesic neighbourhood (green) of a point (red)

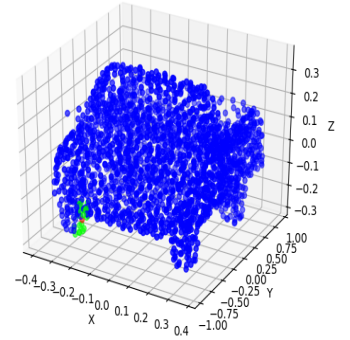
Figure 3.1: Comparison between Euclidean(Fig b) and Geodesic(Fig c) neighbouhood of a point. Euclidean neighbourhood includes points from disconnected bench while geodesic neighbourhood considers points only from the connected parts in the bench



(a) Surface Plot Of car



(b) Point cloud (blue) of a car showing Euclidean neighbourhood (green) of a point (red)



(c) Point cloud (blue) of a car showing Geodesic neighbourhood (green) of a point (red)

Figure 3.2: Comparison between Euclidean(Fig b) and Geodesic(Fig c) neighbouhood of a point in the tyre of the car. Euclidean neighbourhood includes points from front hood of the car as well while geodesic neighbourhood considers points only from the tyre part and doesn't extend to hood part which is not directly connected to the tyre.

3.2 Geodesic Distance And Geodesic Neighbourhood

Lantuéjoul and Maisonneuve (1984) define Geodesic arc as follows, *"Consider an arbitrarily shaped object X and let x and y be two points within X. There exist several paths in X linking x and y. The shortest one is called the '*geodesic arc*'(See Fig.3.3) and its length is denoted by $d_x(x, y)$"*

In the context of our problem, Geodesic distance between two points on the surface of an object is the length of the shortest path connecting the points which lies along the surface of the object. For 3D models, we represent them by graphs in 3D. The geodesic distance is thus calculated as the shortest path from one point to other in the graph. Thus it becomes a shortest path problem in three dimension for which we can use Dijkstra's algorithm (Dijkstra *et al.* (1959)), A-star search (Hart *et al.* (1968)) etc.

The geodesic distance-based local neighbourhood of a point contains all of the points that lie within a certain geodesic distance T_g from the point. The set of geodesic distance-based neighbourhood points P_{pg} of a point p is determined according to Equation 3.1, where $||p_1 - p_2||_g$ represents the geodesic distance between points p_1 and p_2. T_g is the geodesic distance threshold for neighbourhood points and N is the number of points in the point cloud.

$$P_{pg} = \{p_i : ||p - p_i||_g < T_g\} \qquad \forall i = 1, 2, ...N \tag{3.1}$$

Figure 3.3: (1) Paths within the object (2) Geodesic arc.Lantuéjoul and Maisonneuve (1984)

Now, given we have got the intuition behind the geodesic distance and neighbou-

hood, we shall discuss the proposed architecture in the next section.

3.3 magNet Architecture

3.3.1 Problem Statement

Consider a discrete metric space $\phi = (P, G, d)$ whose metric is inherited from an undelying Geodesic space R^3_d, where $P \subseteq R^3$ is the set of points, G is a underlying connected graph structure of P and d is the distance metric(Geodesic). We are interested in learning set functions f that take such ϕ as the input and produce information regarding ϕ. In practice, such f can be classification function that assigns a label to ϕ or a part segmentation function that assigns a per part label to each member of P .

3.3.2 Method

In this subsection, we shall discuss about the proposed architecture in detail. **magNet** consists of an **abstraction module** at its heart. The task of an abstraction module is to sample, group and perform convolution operation on point features. It consists of three steps. 1) Sampling(3.3.3) 2) Grouping(3.3.4) 3) Convolution(3.3.5). We discuss about each of the three steps in detail below.

The **abstraction module** is applied hierarchically. The intuition is to under the connected components of the model at multiple scales. At initial levels, the network will try to learn the underlying strcture at more local levels. With increasing abstraction, the FoV increases and network will try to understand the structure at larger scale. This multiple levels of abstraction makes the netowrk to look at finer as well as coarse details in the point cloud structure and learn effectively.

3.3.3 Sampling

We use the sampling technique used in PointNet++. Farthest Point Sampling(FPS) samples points such that the coverage is maximum. Given input points $\{x_1, x_2, ..., x_N\}$, iterative FPS is applied to choose the subset $\{x_{i_1}, x_{i_2}, ..., x_{i_{N'}}\}$, such that x_{i_j} is the most dis-

tant point(in euclidean space) from the set $\{x_{i_1}, x_{i_2}, ..., x_{i_{j-1}}\}$ with regard to rest points.

Sampling layer takes the point cloud P_{us} of dimension $(N, 3)$ as the input and performs FPS in its first stage and generates farthest point indices $Idx_{fps}(N')$. The second stage is a gathering stage which gathers the points at indices Idx_{fps} from P_{us} and generates the sampled points cloud P_s of dimension $(N', 3)$.(See Fig. 3.4).

The pseudocode for FPS is shown in Algo. 1.

Algorithm 1: FPS algorithm

Data: Input Point Set C, Number of output points N'
Result: Sampled Point Set C_s
initialization $S = \{$ Random Point from $= C\}$;
while $size(S)$!= N' **do**
    get p = farthest point in C from S;
    insert p in S;
end
assign C_s = S;

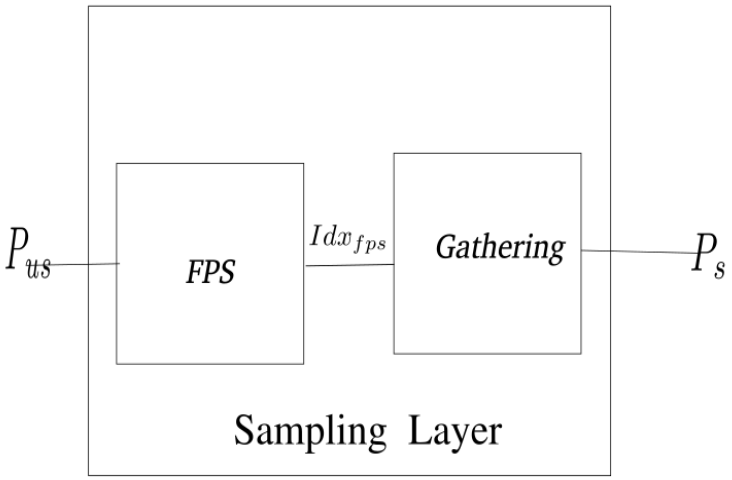
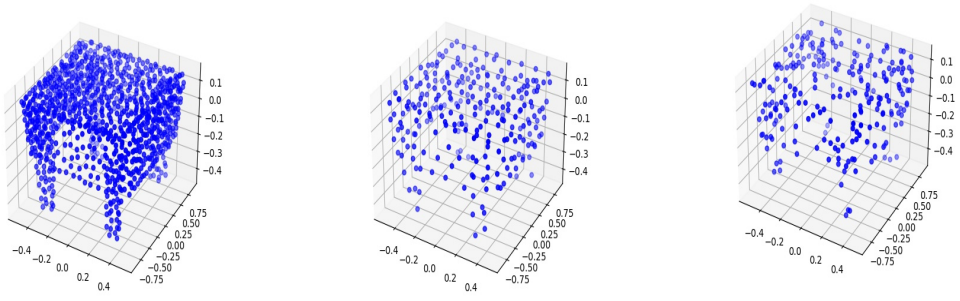


Figure 3.4: Sampling Layer

We can clearly observe in Fig. 3.5, the structure of the point cloud is more clear in FPS than in uniform sampling scenario.



(a) Point Cloud with 1024 points

(b) Point Cloud with 256 points(FPS)

(c) Point Cloud with 256 points(Random Uniform Sampling)

Figure 3.5: FPS applied on a point cloud of 1024 points resulting in a point cloud of 256 points. The coverage of points in case of FPS is clearly better than Random uniform sampling

3.3.4 Grouping

The major contribution of our work lies in this step. As the PointNet++ architecture uses Euclidean neighbourhood, in our work, we use Geodesic neighbourhood instead. The grouping module takes the pre-sampled point cloud, sampled point cloud and corresponding 3D scanned model data. The geodesic neighbouhood for a point set is calculated with reference of its respective high resolution scan.

The scan data needed as input for grouping layer includes the traingular face data of 3D models, the graph generated from the 3D model with the graph weights and connections. Along with that, the vertices from the high resolution scan(not necessarily same as the point cloud) are also needed. The grouping model has two stages. First stage generates the indices of the geo-neighbours for each point. Second stage groups the point features of the neighbourhood points for each sampled point. In short, the output of this layer is the feature vector of geodesic neighbourhood for each sampled point, neighbourhood taken from pre-sampled(high resolution) point cloud.

Fig. 3.6 illustrates the grouping layer.The input to the first stage is unsampled point set P_{us} with size $(N, 3)$, sampled point set P_s of dimension $(N', 3)$ and the 3D model data md. First stage in this layer actually calculates the indices of geodesic neighbouhood of each point in P_s from P_{us}, Id_{geo} of size (N', K), where K is the number of

geodesic neighbourhood for each point and is a hyperparameter. More details about calculating neighbourhood can be found in Appendix B. The second stage takes in Id_{geo}, P_{us}, P_s and feature vector of unsampled point set $f_{us}((N, C))$ and generates the grouped feature f_{geo_s} of size $(N', K, C + 3))$, which is passed to the convolution layer.

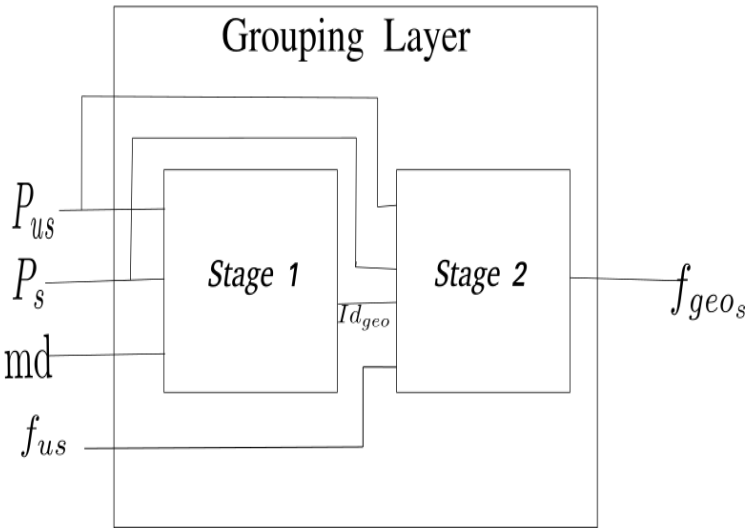


Figure 3.6: Grouping Layer

Fig.3.7 shows geodesic neighbouhood calculated for a point in the point cloud of a chair. We can observe that the nearest points on the outer face of the model are captured as the geodesic neighbourhood of the point under observation.

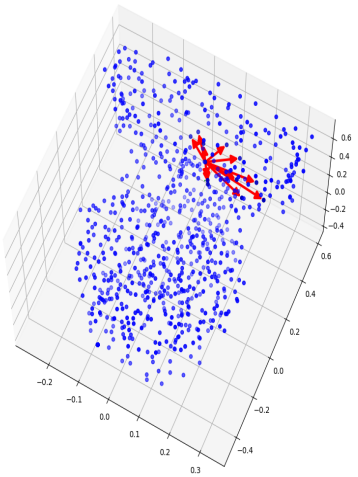


Figure 3.7: Geodesic neighbouhood of the a point inside the point cloud

3.3.5 Convolution Layer

The convolution layer(See Fig. 3.8) consists of multiple *convolution blocks*. Each *convolution block* consists of a 2D convolution layer followed by BatchNormalization(Ioffe and Szegedy (2015)) and a ReLU layer. t is a architecure parameter which decides how many convolutional blocks to be used in a convolution layer of magnet. We use a maxPool layer at the end which outputs the point cloud feature(at that abstraction level) f_s.

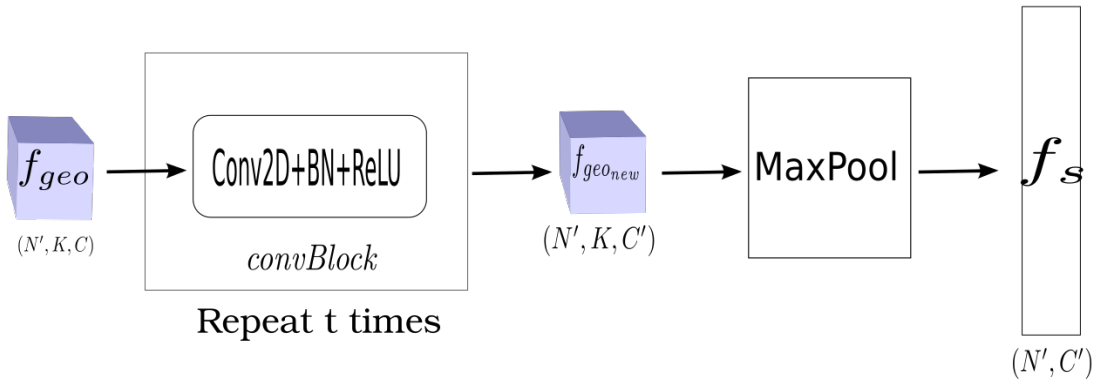


Figure 3.8: Convolution Layer

Fig 3.9 shows theabstraction module end to end with necessary connections.

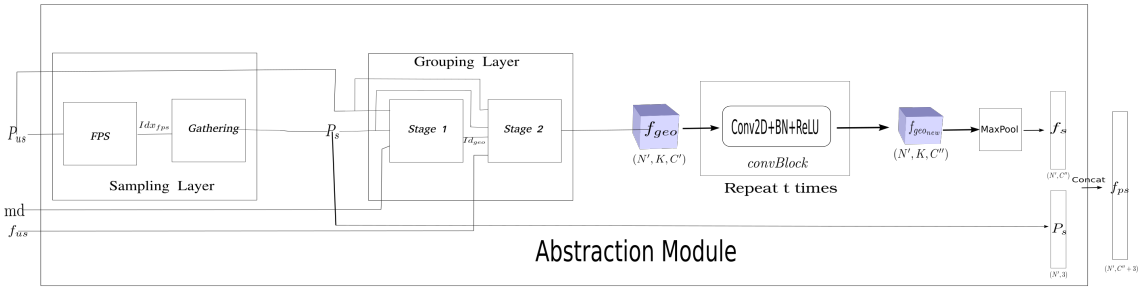


Figure 3.9: Abstraction Module

We refer abstraction module as $AM(K, N', dim)$ with dim being the output channels of final $convBlock$, K as the number of geodesic neighbours for the module and N' as the number of sampled point. In such a framework, the **magNet** architecture is

shown in Fig 3.10. N is the number of input points in the point cloud. The input model data is not explicitly shown here for better clarity of the figure but one should take into account its presence for the functioning of this model. The model generates the feature for the entire point cloud f_{pc} of size $(1, 1024)$.

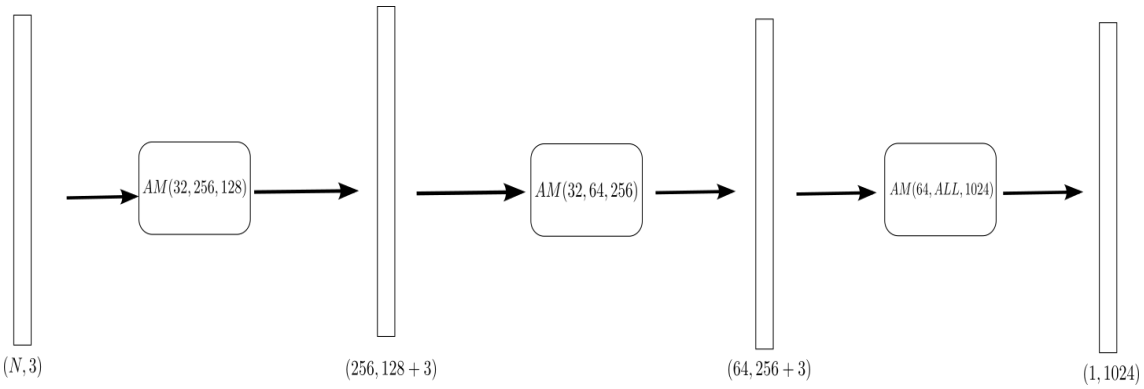


Figure 3.10: magNet Architecure

3.3.6 magNet for classification

We further take the point cloud feature f_{pc} for classification. The feature vector goes through fully connected layer followed by BatchNorm and a ReLU layer. Fig 3.11 illustrates the classification head of magNet. $Linear(dim)$ represents Fully Connected Layer $FC(inputDim, dim)$ followed by BatchNorm layer $BN(dim)$ and a $ReLU$ layer. Final FC layer returns a vector of size 40 which is the number of classes for ModelNet40 Wu *et al.* (2015) dataset.

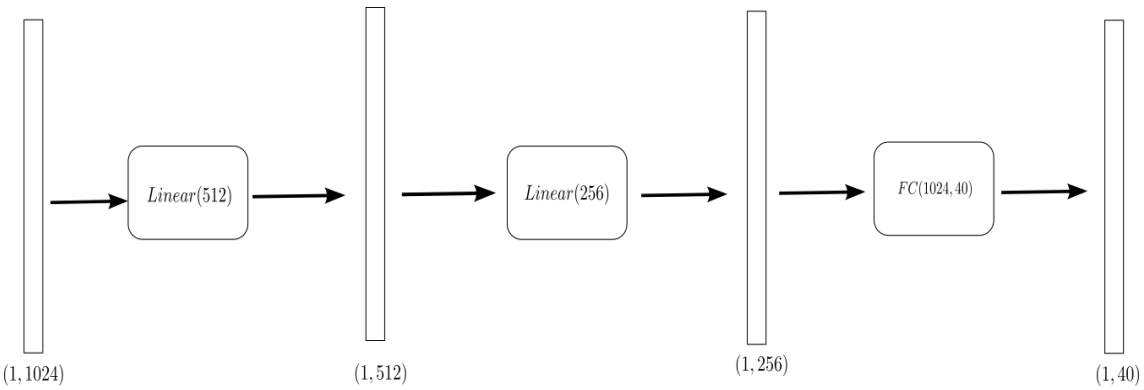


Figure 3.11: magNet for Classification

3.3.7 magNet for part segmentation

In the abstraction modules, we subsampled the original point set. For part segmenation we need to up-sample the feature set segmentation task such as semantic point labeling, we want to obtain point features for all the original points. The primitive solution is to always sample all points as centroids in all set abstraction levels, which however results in high computation cost. A smarted way that fits in a learning system is to propagate point features from the sampled set to the original point set. Quoting from Qi *et al.* (2017*b*),

"We adopt a hierarchical propagation strategy with distance based interpolation and across level skip links. In a feature propagation level, we propagate point features from $N_l \times (d+C)$ points to N_{l-1} points where N_{l-1} and N_l (with $N_l \leq N_{l-1}$) are point set size of input and output of set abstraction level l.We achieve feature propagation by interpolating feature values f of N_l points at coordinates of the N_{l-1} points. Among the many choices for interpolation, we use inverse distance weighted average based on k nearest neighbors."

The interpolated features are concatened with skip connected features from the respective abstraction module output. The concatenated features are passed through the convolution layer(Refer Section 3.3.5). The feature vector of the point set is updated due to the convolution layer. The process is repeated and the propagated point feature is expanded until the propagation reaches the original point feature. The inverse distance based weight(Eqn. 3.2) gives higher priority to the points which are closed in the Metric space(Euclidean in this case)

$$f^j(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i^j}{\sum_{i=1}^{k} w_i(x)} \tag{3.2}$$

where

$$w_i(x) = \frac{1}{d(x, x_i)^p}, j = 1, ..., C \tag{3.3}$$

In default case $p = 2$ and $k = 3$. The segmentation head can be seen in Fig. 2.5.

In this chapter, we discussed in detail about our proposed architecure which we name as **magNet**. We understood why geodesic neighbourhood is effective and how

can we utilize this while working with 3D point clouds. In the next chapter, we shall provide details about our training method and the results we obtained.

CHAPTER 4

Experimental Results

We trained the model using in **PyTorch Lightning** which is a lighweight PyTorch wrapper with high level abstraction over PyTorch functions. In this chapter, we provide details about the preprocessing and training step and provide the results that we obtained.

In Section 4.1, we briefly disucss about the Dataset and the proprocessing step. We shall discuss about the classification problem in Section 4.2 In Section 4.3, we shall overview the Segmentation problem.

4.1 Dataset description and preprocessing

We evaluate the performance of our proposed neural network models on the object classification and part segmentation tasks using ModelNet40 and ShapeNet part datasets respectively. ModelNet40 consists of CAD models of objects with class labels provided for each of the object. ModelNet40 dataset consists of 12,311 CAD models of objects categorised into 40 classes. The dataset is further split into training and test sets, each of which contain 9843 and 2468 number of samples respectively. ShapeNet part dataset consists of 16881 CAD models of objects which are divided into training test and validation sets, each of which contain 12137, 2874 and 1870 number of object CAD models respectively. The objects in this dataset are divided into 16 categories of objects. This dataset provides point clouds of the objects which consists of points sampled from the surface of the CAD models. Each point in the point cloud is labelled with a part labels corresponding to the part of the object. Each category of the object has 2-6 part labels,with a total of 50 part labels in the ShapeNet part dataset.

The following data preprocessing method is taken from MS thesis of Nagender V S S(Dept. of CS, IIT Madras, 2019). Original work by him involved doing this step in Python using ***scipy*** library and is a dedicated preprocessing step. But in our work, we developed **custom cuda kernels from scratch** that calculate the geodesic neighbourhood during network. But still as the process of finding the geodesic neighbouhood is

same we present it in this section itself. Creating the graph structure from the 3D model is a preprocessing step in our work and is mentioned in Appendix A.

Qi *et al.* (2017*a*) uniformly sampled 1024 points from the surface of the CAD models of objects and normalized them into a unit sphere. In the case of ShapeNet part dataset, Qi *et al.* (2017*a*) have sampled 2048 points per point cloud in the train and validation sets of the dataset. In the case of test samples, they use all of the points provided for each point cloud in the ShapeNet part dataset for testing. For the sake of comparison, we use the same sets of point clouds of objects used by Qi *et al.* (2017*a*) in both of the datasets for training and testing our neural network architectures.

In addition to point clouds of objects, we also need geodesic distances between points in our work. Geodesic distance between two points is the distance be- tween two points that is measured along the surface of the object. To compute geodesic distance between points, we use the CAD models of the objects provided in the dataset. The surfaces in the CAD models of objects are represented using triangular meshes. Each triangular mesh consists of vertices and faces, where each face is represented by the three vertices forming the corresponding triangular face. We convert the triangular mesh into a graph containing vertices and edges, where edges are the sides of the triangular faces. The 3D coordinates of the vertices in the graph are normalised into a unit sphere using the same parameters (mean and scaling factor) that used to normalise the corresponding point cloud of the object into a unit sphere. The weight of each edge is the distance between the normalised points on either side of the edge. For each point in the point cloud of the object, we find the nearest vertex in the corresponding graph of the object. We approximate geodesic distance between two points as the shortest path connecting their corresponding nearest vertices in the graph of the object. We compute shortest path between vertices of the graph using Dijkstra's shortest path algorithm.

For the Dijkstra's shortest path algorithm to compute shortest path connecting any two points in a graph, the input graph should form one connected component. Due to discrepancies in the mesh representation and due to representing different parts of the objects with disconnected triangular meshes, the graph of the object constructed from the triangular mesh does not form one connected component. In our work, we preprocess the triangular mesh such that most of the mesh discrepancies resulting in disconnected components are corrected. After removing the discrepancies, if there are

any disconnected components in the resulting graph, we find the vertices that are within a certain distance threshold in each pair of disconnected components and add edges connecting those vertices. This results in a graph that forms one connected component. The method adopted to construct a graph that forms one component is described in Appendix A.

4.2 Classification

In this section, we describe the experiments conducted to evaluate the proposed object classification network described in Section 3.3.6. We train the network using the train set of ModelNet40 dataset and report results on its corresponding test set. The evaluation measures of classification task on this dataset are the overall accuracy and mean accuracy across classes. We evaluate the effectiveness of our proposed network by comparing the performance with the similar PointNet, PointNet++ and LSNET proposed by Nagender et al.(2019, IIT Madras).

4.2.1 Implementation Details

In our implementation, we perform data augmentations on the training sample. We perform random scaling, random rotation, random perturbed rotations, Jitter, Global random translation and Random Point dropout. No augemetations were done with the validation set.

We created the network in PyTorch(with PyTorch Lightning as wrapper). As we had used custom cuda kernels that supported only array pointers(and not highly abstract data types like CSR matrices etc.), we needed to do extra preprocessing on the graph elements in order to pass them to the custom kernel. More about that in Appendix B The input point cloud P_{us} and the model data md are created from the data loader and are clubbed together(across the batch) using a custom collate function. The input point coordinates are spherically normalized and have a value in the range [-1, 1]. We consider the local neighbourhood distance threshold T_g as a hyperparameter. Table 4.1 shows the hyperparameter setting for the network.

During training, we use the softmax cross entropy function to compute loss and

Table 4.1: HyperParameter setting for the classification task.

AM	N'	$T_g(threshold)$	K
1	256	0.2	32
2	64	0.4	32
3	GroupAll	GroupAll	ALL

adam optimizer to update the weights based on gradients. We train our network with an initial learning rate of $1e-4$ which gets multipled by a factor of 0.8 every 5000 training batches. We use a weight decay of $1e-5$. We also use a Dropout of 0.1-0.2 for Conv2D layers and 0.4-0.5 for Linear Layers in the network.

For the classification task, we compute the point cloud feature f_{pc_a} at every abstraction level as shown in Fig. 3.10. The final feature vector representing entire point cloud f_{pc} passes through the classification head of the network and generates a vector equal to the number of output classes. The classlabel is determined based on Equation 4.1. The classification architecture is shown in Fig 3.11. We use two $Linear$ modules(Refer Section 3.3.6) with output features of 512 and 256. The output of final linear layer passes through the FC layer.

$$classLabel = \arg \max_{1 \leq i \leq NC} out[i] \tag{4.1}$$

We can see the comparison between the networks in Classification accuracy in Table 4.2. We can observe despite proposed theoretical robustness, **magNet** couldn't perform well. We attribute few reasons to this below:

- We couldn't find the right training setting for the model as the model was getting overfitted after certain number of epochs
- Another issue is attributed to the memory requirements for the cuda kernel. For each point, we could construct a pre-specified size of graph(450 in this case). This is due to memory constraints for the GPU. But we observed that for few points the number of graph points needed exceeds 450 and in that case we needed to put some hard conditions on the kernel. This causes certain discrepancy in the shortest path distance calculation in dijkstra and gives insufficient answer i.e. number of neighbourhood points and network performs poorly such a scenario, which we believe is happening in this case.

Table 4.2: Classification Accuracy

Model	Accuracy	Mean Accuracy
PointNet	89.2	86.2
PointNet++	90.7	-
LsNet(Geodesic Neighbourhood)	92.1	89.4
magNet	86.7	82.4

4.3 Part Segmentation

In this section, we describe the architecture for Part Segmentation task. In this section, we describe training details and results we obtained.

The point cloud feature vector f_{pc} is passed through the Feature Propagation module(Refer to Section 3.3.7). Each feature propagation(FP) module has a convolution Layer which updates the point features. The FP module is repeated till the point features of the entire input point cloud are propagated. Table 4.3 shows the hyperparameter setting for the Segmentation Architecture.

We use softmax cross entropy loss. The output of segmentation network is a $(2048, 50)$ dimension matrix(There are 50 part segmentations of 16 classes of objects in ShapeNet part dataset). The predicted part segmentation label is then calculated by performing the $\arg max$ operation on the output(Eqn. 4.2)

$$partLabel[i] = \arg \max_{1 \leq j \leq 50} out[i][j], \forall i \in \{0, 1, 2..., 2047\} \tag{4.2}$$

We trained the network with LR of 1e-3 and multipled by 0.8 after 3000 training batches. We present the results after 20 epochs of training in table 4.4

AM	N'	$T_g(threshold)$	K	MLP
1	512	0.2	32	[3, 64, 64, 128]
2	128	0.4	32	[128, 128, 128, 256]
3	32	0.8	32	[256, 256, 512, 1024]

(a) For the Abstraction Module

FP	MLP
1	[1024+256, 256, 256]
2	[256+128, 256, 128]
3	[128+3, 128, 128]

(b) For the Feature Propagation Module

Table 4.3: HyperParameter setting for the Segmentation task.

Table 4.4: Segmentation Results

Class	mIOU
Airplane	66.6
Bag	63.96
Cap	75.4
Car	63.5
Chair	84.75
Earphone	69.08
Guitar	83.84
Knife	76.67
Lamp	71.03
Laptop	93.18
Motorbike	26.46
Mug	89.86
Pistol	63.41
Rocket	41.85
SkateBoard	62.08
Table	77.81
mean mIOU(all shapes)	76.05

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this report, we have proposed a Multi Abstraction Neural Network for the task of Point Cloud understanding. The motivation behind our idea was the significant improvement in performance shown by convolutional neural networks on computer vision tasks with images as input. But the input to a sequence of convolution layers is either a 2D grid of pixels (images) or a 3D grid of voxels. In our proposed model, we take the entire point cloud as the input and systematically sample and process the point set in order to get more and more abstract point features. The motivation was taken from the way 2D CNN models work with increasing Field of View as the convolutional layers are applied to 2D feature map of the image. At each abstraction layer, our model learns an underlying local shape feature formed by the connected points. With increasing abstraction, the field of view is enlarged and the model looks at bigger and bigger underlying shape. This theoretical intuition was the motivation behind this architecture. We used geodesic distance based neighbourhood in order to extract the neighbourhood for a point. We observed that in addition to being translation invariant, the geodesic neighbourhood-based point feature is also invariant to articulations around joints of the object.

We created custom cuda kernels from scratch for the process of calculating geodesic neighbourhood for the points. We have created a **custom PyTorch extension** for the same. Anyone willing to experiment with the extension will be able to do so. This provides a better streamlining for any future work. Unfortunately we couldn't achieve expected results in the classification task reasons for which we have mentioned in this report. Along with classification, we have presented the part segmentation network in our report, but we couldn't train the model due to technical issues. The code base is available in open source and anyone can perform experimentations on it and attempt to get even better results.

Despite not being able to achieve satisfactory results, we understood that Geodesic neighbourhood is an effective mechanism to represent a point feature in 3D space. The

point feature is robust to translation, rotation and joint articulation variant. We provide the researchers community with a robust PyTorch extension that calculates the geodesic distance and neighbourhood for a point in a graph which is another learning problem in itself.

5.0.1 Future Work

One clear cut observation from our study was regarding overfitting of the model. There is an opportunity to build a better performing NN network using the robust sampling+grouping method we provide. This can provide us state of the art results. Another opportunity for future work lies in modifying the cuda kernels to eliminate the restriction for pre defined number of graph points. This shall provide us with even better neighbourhood for the points. Due to lack of sufficient time, we couldn't perform enough experiments with varying r(radius threhshold) and K(Number of neighbourhood points per point). A better set of hyperparameters can definitely provide us better results for our setting.

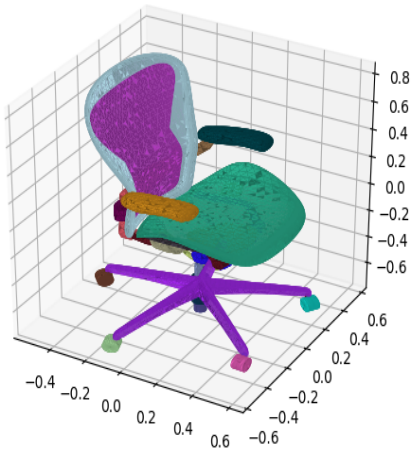
Another possibility that builds up is to find the connected components in the model directly. Currently we are finding the connected components only along surface of the 3D model. We can port this problem as a pose detection problem and find the connected components of the model and build up on that. We can use similar multiple abstraction layer which looks at actual connected components of the 3D model at multiple levels. This can provide even better feature representation for the point cloud. The **PyTorch Extension** that we provide which forms the very heart of our work can be elevated and used for multiple other tasks including but not limited to Pose Detection, 3D Semantic Segmentation etc. It ultimately ends up on researcher's idea and creativity to do further research using the tool we provide.

APPENDIX A

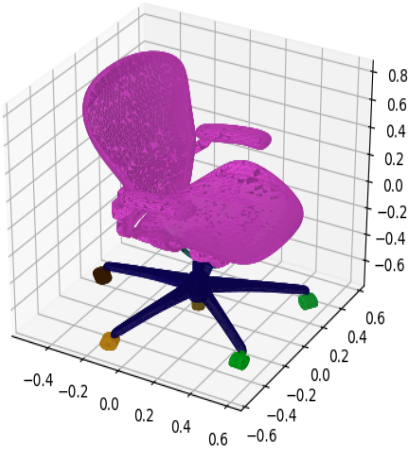
Graph representation of objects constructed from their corresponding CAD models

In this chapter, we describe the method we use to construct a graph representing an object from its corresponding triangular mesh. The graph resulting from this approach consists of as many connected components as the number of disconnected parts in the object. Most of the objects in the dataset are such that their corresponding parts form one connected object. The graphs corresponding to these objects which are generated using the approach presented in this section should have only one connected component.

The triangular mesh of an object might contain discrepancies such as isolated vertices, duplicated vertices and self intersecting submeshes. These discrepancies might arise due to the software that is generating these triangular meshes or due to some other manual errors in CAD model generation. Isolated vertices in a mesh are the vertices that are not part of any of the triangular faces. The first step in our approach is to remove all of the isolated vertices and update the vertex indices in the list of triangular faces that reflect this change. Duplicated vertices are vertices which have the same 3D coordinates. Duplicated vertices might result in disconnected triangular meshes if two or more disconnected meshes use the same set of 3D coordinates but with a different vertex index. The next step in our approach is to remove duplicated vertices by assigning a single index to each set of duplicated vertices and update the vertex indices in the list of triangular faces that reflect this change. Self intersecting submeshes are submeshes that are spatially intersecting with each other but are disconnected as there no common vertices (and faces) where the meshes intersect. Such self intersecting meshes are usually a result of combining two CAD models of parts of object to form the one single part, but the part CAD models are still represented using their individual surface representations. These self intersections can be resolved by adding more vertices and faces that connect the intersecting parts. Most of the surfaces of the parts of the object become connected after removing the three discrepancies. In our implementation, we use functions provided by PyMesh library to resolve these three discrepancies.



(a) Disconnected parts of a chair with 41 disconnected parts.



(b) Parts of the chair after removing isolated vertices, duplicated vertices and resolving self intersection. Number of disconnected parts in this figure is 10.

Figure A.1: Figure a shows a chair taken from an unprocessed triangular mesh. Figure b shows the parts of the chair after removing isolated vertices, duplicated vertices and resolving self intersections. It can be seen that most of the parts of the chair are connected after removing the three discrepancies in the triangular mesh of the chair.

After resolving discrepancies in the triangular mesh, the 3D coordinates of all of the vertices are normalized into a unit sphere using the same parameters (mean and scaling factor) that are used to normalize the 3D coordinates of points in the corresponding point cloud. In some of the objects, if the part of an object consists of a plane surface, the corresponding triangular mesh contains triangular faces with long edges. For our purpose of approximating geodesic distance between points in a point cloud as the shortest distance between their corresponding nearest vertices, we need a triangular mesh with high resolution even on plane surface. In order to generate such a triangular mesh with high resolution, we split all of the long edges in the mesh into smaller ones and add corresponding vertices and edges to the mesh. In our current implementation, we allow a maximum edge length of 0.05 and split any edge longer than this length into smaller edges.

In the next step, if there are any disconnected parts present in the object, we find distances between the vertices corresponding to each pair of disconnected parts. If any pair of disconnected components have vertices which are within a certain distance threshold, we add edges between all of those vertices, thus connecting that pair of disconnected components. In our implementation we use a distance threshold of 0.1 (twice the maximum edge length in the high resolution triangular mesh generated in the previous step) to connect vertices across disconnected parts with edges. In the final step, we generate a graph from the processed triangular mesh of the object. The vertices of the triangular mesh become the vertices of the graph and the sides of the triangular faces and the edges connecting vertices across disconnected components become edges of the graph.

The high resolution graph with one connected component (in most of the objects) generated in the final step is used to determine geodesic distances between points in the point cloud of an object.

APPENDIX B

Finding Geodesic Neighbourhood Using Model Data

In this chapter, we shall discuss about method to find the geodesic neighbourhood of a point belonging to the sampled point set P_s, from the unsampled point set P_{us}(point is P_s and neighbourhood from P_{us}. As discussed in Appendix A, we now have a constructed graph and face data of the 3D model. Using that we proceed to create the geo-neighbourhood.

In the preprocessing, we extract the number of points in the graph. We extract the Index Pointer for the graph. We extract the beginning pointers for the the graph vertex array, faces data and extracted graph data across a batchSize. As the number of graph vertices is variable across samples, we need to concatenate this data and need to extract the beginning pointers for each sample in order to process them in a batch. The concept of Indptr for a csr is beautifully illustrated in Fig. B.1. The data and indices represent the non-zero data and indices respectively in the graph.

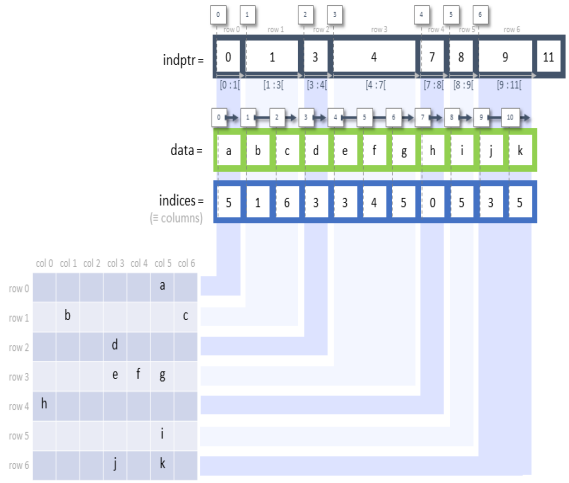


Figure B.1: IndPtr for a CSR graph IMGUR

After doing this pre-processing step we pass this through a custom collate function that appropriately clubs together the data across the batches and passes through to the cuda kernel via a Torch c++ wrapper.

Inside the cuda kernel, we perform the actual computation parallelized across the batch. For each point p in the sampled space, we first calculate the nearest face point using euclidean distance. The face data is provided as the input. Now the computation revolves around this nearest face point p_{nfp} for the current point in sampled space. Once that is done, we calculate the nearest graph points to the current point using euclidean distance metric same as the threshold r used in the respective abstraction module. Here as we can not go on take an un-specified number of points as is the case, we restrict ourselves to 450 **nearest** points which are within the r threshold.

The nearest graph points set indices $\{i_{ng}\}$ is used to build the subgraph for the point p on which we apply dijkstra. As these point indices are nothing but the indices of the rows of the original graph, we select those rows from graph and calculate the subgraph of size $(450, 450)$ in a smart way. We perform dijkstra's shortest path algorithmDijkstra *et al.* (1959) on this point set with p_{nfp} as the source point and find the shortest path distance d_{ng}. One thing to note here is all the points selected in the graph are not necessarily points from the face but include points from inside parts of model. Hence once the shortest distances along the graph is calculated, we need to find only those specific points which form a part of the face of the model and hence make the geodesic neighbourhood of the point p.

For that, we perform an Euclidean search on the P_{us} for point p. All the points in P_{us} that lie within the threshold r(same as r for the abstraction module) are found. For all those points, we find the nearest face point. If the index of nearest face point lies in the $\{i_{ng}\}$ set and its distance $d_{ng} \in \{d_{ng}\}$ is less that r, then the respective point in P_{us} is classified as the geodesic neighbour of the point p. This process in parallely performed for all N' points across b samples in a batch and the geodesic neighbourhood set of size $(b, N', K, 3)$ is calculated, where K is the maximum number of neighbourhood points selected.

REFERENCES

1. **Arras, K. O.**, **O. M. Mozos**, and **W. Burgard**, Using boosted features for the detection of people in 2d range data. *In Proceedings 2007 IEEE international conference on robotics and automation*. IEEE, 2007.

2. **Atzmon, M.**, **H. Maron**, and **Y. Lipman** (2018). Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*.

3. **Ben-Shabat, Y.**, **M. Lindenbaum**, and **A. Fischer** (2018). 3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*, **3**(4), 3145–3152.

4. **Bogdan Rusu, R.**, **A. Sundaresan**, **B. Morisset**, **K. Hauser**, **M. Agrawal**, **J.-C. Latombe**, and **M. Beetz** (2009). Leaving flatland: Efficient real-time three-dimensional perception and motion planning. *Journal of Field Robotics*, **26**(10), 841–862.

5. **Chen, J.**, **B. Lei**, **Q. Song**, **H. Ying**, **D. Z. Chen**, and **J. Wu**, A hierarchical graph network for 3d object detection on point clouds. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

6. **Dijkstra, E. W.** *et al.* (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, **1**(1), 269–271.

7. **Graham, B.**, **M. Engelcke**, and **L. Van Der Maaten**, 3d semantic segmentation with submanifold sparse convolutional networks. *In Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

8. **Hart, P. E.**, **N. J. Nilsson**, and **B. Raphael** (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, **4**(2), 100–107.

9. **Ioffe, S.** and **C. Szegedy**, Batch normalization: Accelerating deep network training by reducing internal covariate shift. *In International conference on machine learning*. PMLR, 2015.

10. **Kanezaki, A.**, **Y. Matsushita**, and **Y. Nishida**, 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.

11. **Klokov, R.** and **V. Lempitsky**, Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. *In Proceedings of the IEEE International Conference on Computer Vision*. 2017.

12. **Lantuéjoul, C.** and **F. Maisonneuve** (1984). Geodesic methods in quantitative image analysis. *Pattern recognition*, **17**(2), 177–187.

13. **Li, C.-L.**, **M. Zaheer**, **Y. Zhang**, **B. Poczos**, and **R. Salakhutdinov** (2018). Point cloud gan. *arXiv preprint arXiv:1810.05795*.

14. **Lin, Z.-H.**, **S.-Y. Huang**, and **Y.-C. F. Wang**, Convolution in the cloud: Learning deformable kernels in 3d graph convolution networks for point cloud analysis. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

15. **Liu, Y.**, **B. Fan**, **S. Xiang**, and **C. Pan**, Relation-shape convolutional neural network for point cloud analysis. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.

16. **Maturana, D.** and **S. Scherer**, Voxnet: A 3d convolutional neural network for real-time object recognition. *In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015.

17. **Moenning, C.** and **N. A. Dodgson**, Fast Marching farthest point sampling. *In Eurographics 2003 - Posters*. Eurographics Association, 2003. ISSN 1017-4656.

18. **Navarro-Serment, L. E.**, **C. Mertz**, and **M. Hebert** (2010). Pedestrian detection and tracking using three-dimensional ladar data. *The International Journal of Robotics Research*, **29**(12), 1516–1528.

19. **Qi, C. R.**, **H. Su**, **K. Mo**, and **L. J. Guibas**, Pointnet: Deep learning on point sets for 3d classification and segmentation. *In Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017*a*.

20. **Qi, C. R.**, **L. Yi**, **H. Su**, and **L. J. Guibas** (2017*b*). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*.

21. **Riegler, G.**, **A. Osman Ulusoy**, and **A. Geiger**, Octnet: Learning deep 3d representations at high resolutions. *In Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

22. **Shi, S.**, **Z. Wang**, **J. Shi**, **X. Wang**, and **H. Li** (2020). From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *IEEE transactions on pattern analysis and machine intelligence*.

23. **Simonovsky, M.** and **N. Komodakis**, Dynamic edge-conditioned filters in convolutional neural networks on graphs. *In Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

24. **Uy, M. A.**, **Q.-H. Pham**, **B.-S. Hua**, **T. Nguyen**, and **S.-K. Yeung**, 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.

25. **Wu, Z.**, **S. Song**, **A. Khosla**, **F. Yu**, **L. Zhang**, **X. Tang**, and **J. Xiao**, 3d shapenets: A deep representation for volumetric shapes. *In Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

26. **Xiang, T.**, **C. Zhang**, **Y. Song**, **J. Yu**, and **W. Cai** (2021). Walk in the cloud: Learning curves for point clouds shape analysis. *arXiv e-prints*, arXiv–2105.

27. **Xu, K.**, **W. Hu**, **J. Leskovec**, and **S. Jegelka** (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

28. **Zaheer, M.**, **S. Kottur**, **S. Ravanbakhsh**, **B. Poczos**, **R. Salakhutdinov**, and **A. Smola** (2017). Deep sets. *arXiv preprint arXiv:1703.06114*.