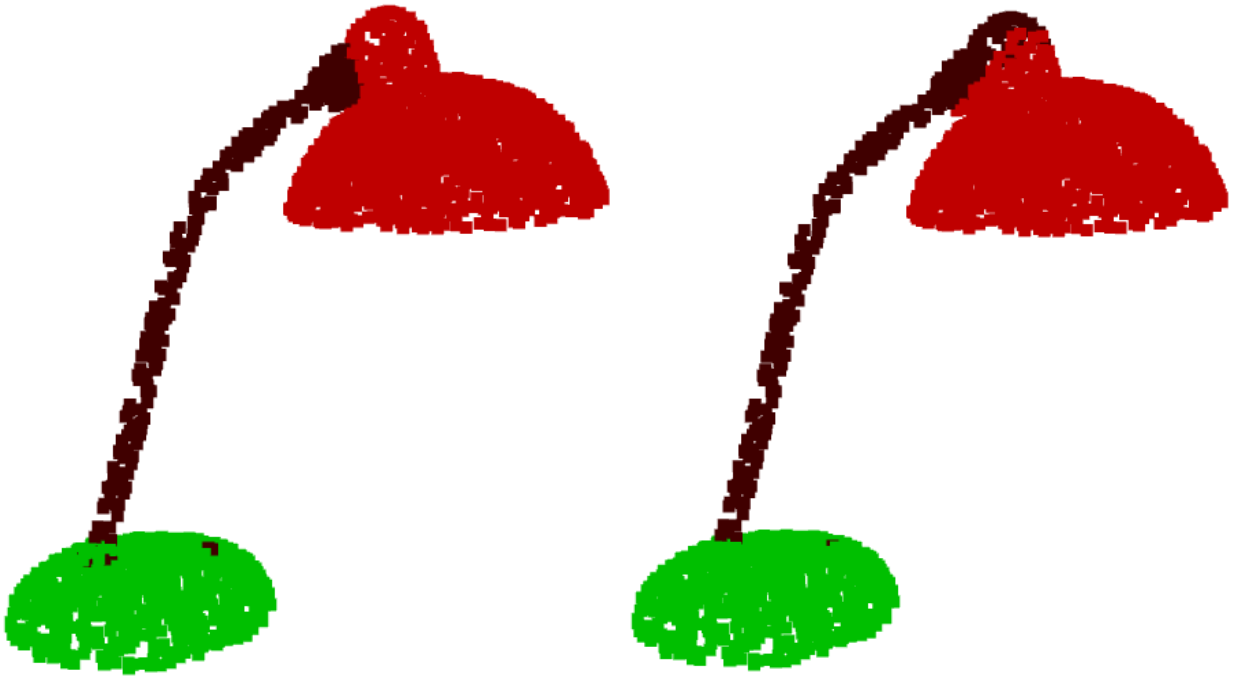


Geometry Identification from Point Cloud Data

Master's thesis in Engineering Mathematics and Computational Sciences

MAITREYA DEEPAK DAVE

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Master's thesis 2021

Geometry Identification from Point Cloud Data

MAITREYA DAVE

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2021

Geometry Identification from Point Cloud Data
MAITREYA DAVE

Supervisor: Ludvig Friborg, Wiretronic AB
Academic Supervisor: Mats Rudemo, Department of Mathematical Sciences
Examiner: Aila Särkkä, Department of Mathematical Sciences

Master's Thesis 2021
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Part segmentation result on point cloud representation of a lamp and Left: Point cloud represented with true labels, Right: Point cloud represented with predicted labels

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Geometry Identification from Point Cloud Data
MAITREYA DEEPAK DAVE
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Abstract: The aim of this thesis is to conduct a comparative study of different algorithms which can learn from 3D (x,y,z) point cloud data and are able to conduct per-point classification. The point clouds of interest for this thesis are the one's which represent a 360-degree view of an object. For example, the input is a point cloud representing a knife with two parts: the blade and the handle. For such an input, the algorithms must be able to classify which points of this point cloud belong to blade and to the handle. To solve this task different types of deep-learning based models like graph-based neural networks and non-grid point convolution networks were implemented and analyzed. These models perform with over 80% accuracy which is quite remarkable given the input is only raw 3D coordinates with no need of voxelization. The models have also been tested on synthetic datasets as well as an object scanned using a LiDAR camera. These algorithms can be applied in autonomous driving, augmented/virtual reality applications, medical data processing and 3D animation industry.

Keywords: point, cloud, part, segmentation, pointnet, dgcnn, kpconv, 3D, deep-learning, supervised

Acknowledgements

I would like to my supervisor Mats Rudemo and examiner Aila Särkkä for their inputs, and guidance throughout the thesis. I would like to also thank Wiretronic AB, for giving me this opportunity and aiding me throughout the thesis. Thank you to both these parties for supporting me throughout this thesis!
Secondly, I would like to thank Chalmers University of Technology for its academic resources and for providing support during these unprecedented times. The courses offered by Chalmers University of Technology have helped me gain valuable experience in both theory and practice. A special thanks to the Chalmers Centre for Computational Science and Engineering (C3SE) for providing me the access to the computer hardware needed to perform this thesis.
Finally, I would like to thank my family and every close friend of mine for their continued unconditional support during the whole process, and for being there all way through everything. Thank you, everyone!

Maitreya Deepak Dave, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

The geometric shape of an object can be described as the description of an object without its scale, orientation and reflection. This means that if we do any of these operations on an object, the geometrical shape or structure of the object should remain the same and not turn into a new distinct shape.
In recent years, research in object identification with point cloud data as an input has turned into a growing topic of research as it leads to many applications in vision perception, virtual/augmented reality, robotics, and medical imaging. The idea of this thesis is to investigate AI-based methods to segment out geometrical shapes from the 3D point clouds with as little human intervention as possible.
Task associated with 3D are

- 3D Geometry Analysis which consists of tasks like object classification, segmentation of a given object or a scene, finding correspondences between different but similar data.
- 3D Synthesis which consists of tasks like reconstruction of a partial point cloud, automatic shape completion or predictive shape modelling.

1.1 Objectives

The objective of this thesis is to conduct a comparative study of different algorithms which segment geometrical shapes from point cloud data. By achieving this, one could use the results for further tasks like component verification, synthetic data generation, visualization and more. The research questions to investigate are as follows

Question 1
What possible deep learning based models exist to perform part segmentation of point clouds? How well do they perform?

Question 2
How efficiently do the models handle raw point clouds as an input ?
Since point cloud data can be obtained either from sensors directly or estimated from images, there can be quite a lot of variation in the point density, missing points, outlier and accuracy of the obtained point cloud. To answer this question, the models are trained on synthetically available data and tested on test dataset which is an unseen subset of the training dataset as well as with point clouds obtained from scanning an object using LiDAR based camera. The input point during

training stage is obtained by uniformly sampling a 3D mesh made in CAD program. However, when testing the point clouds using scanned data, the point cloud has outliers, missing points and randomly jittered points which need to be taken care of.

Question 3

Are all geometric shapes recognized with the same accuracy? Which geometric shapes are easier to identify? The visualizations of predictions made on the test datasets will be analyzed.

1.2 Previous Work

Geometry identification from a point cloud can be seen as a point cloud segmentation task. There has been significant research in this area in recent years, the credit for this advancement goes to machine learning techniques [22, 23, 17, 33]. Methods before machine learning [18, 25] relied on hand-crafted features for specific tasks but it may not always be easy to find the optimal set of features by hand. Deep learning models like [22], showed remarkable results in 3D object classification, part segmentation, and semantic segmentation of point clouds. These results motivated more research in this area, the result of which emergence of models based on 3D point-based networks [17, 31], graph-based networks [36, 34], networks-based on projecting data to a different dimensional space [28, 1], capsule networks [3]. A common approach in all the models has been down-sampling the input point cloud as the cost of computation is directly proportional to the number of input points. Hence, there is an inevitable loss of information. The task of segmentation without downsampling a point cloud is called fine-grained segmentation of point cloud, while [37] delivered very good results for this task it has been approached in a supervised manner which makes it less practical for developing an application with. [9] provides a detailed account of deep-learing based methods for various point cloud related tasks.

1.3 Structure of report

The thesis begins with an introduction of the problem statement, the objective of this thesis, and the previous work done in this area. We start with the background which introduces different types of representations of 3D data and dwells in-depth about point clouds which is the focus of this thesis. Then we provide a basic overview of deep-learning followed by different types of deep-learning approaches taken by researchers for 3D point cloud analysis. Then methodology chapter provides important details of each of the implemented models, explaining how they work. The development stage introduces the hardware setup used to train the models, the hardware used to capture real-world test data, and how this captured data was processed. The results chapter presents numerical and visual results of training and testing the models. The results were examined in the discussion chapter. Finally, the conclusion speaks of what was achieved and proposes possible future work directions.

2
Background

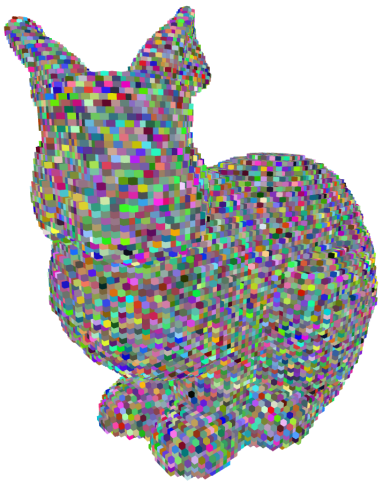
2.1 Introduction to 3D Representations

To represent 3D data, we have multiple representations. Here I briefly introduce a few of the most commonly used representations and then dive deeper into point clouds which is relevant for this thesis.

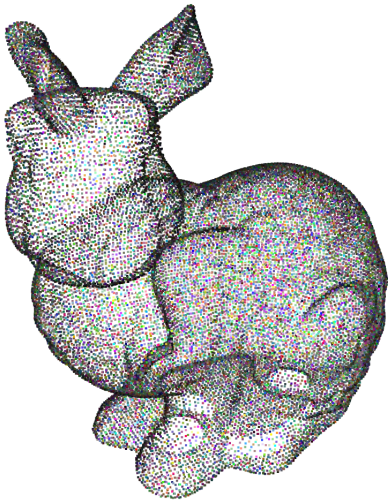
Multi-view representation
A multi-view representation is usually a 2D image based representation method where multiple 2D images of the same object/scene are captured from multiple viewpoints of the 3D object and then processed with 2D convolutional neural network (CNN) models. Usually some kind of localization information about the capture device should be available or estimateable for accurate processing of these images. Multi-view representations are highly convenient because we already have state of art 2D CNN models which are capable of object detection as semantic segmentation. Based on this representation networks like in [29] were developed to learn 3D knowledge of shapes from their 2D views.

Figure 2.1: Image depicting camera taking images of an object from multiple viewpoints

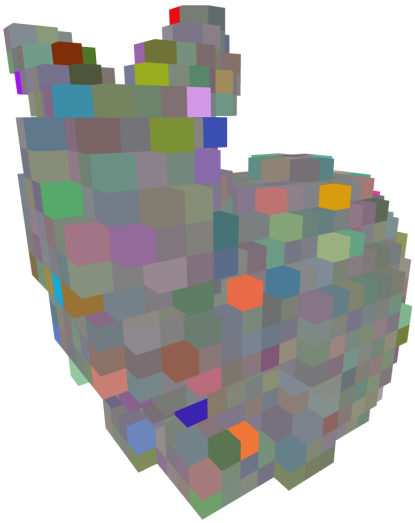
Voxel Grid Representation
Voxel Grid Representation is a Volumetric representation method and the representation is obtained by voxelizing a 3D mesh or 3D point cloud. In context of 3D data, we can start by representing the 3D data with a 3D bounding box, then define a voxel to be a rectangular parallelepiped (whose dimension shall be much smaller than the original bounding box). A 3D grid of such voxels shall be used to match the dimensions of the bounding box. Each voxel has an occupancy value which tells us whether the voxel is within the original 3D data or not. Based on the resolution of our voxel we can have a fine or a coarse representation of our original data.



(a) Voxel Grid of size 0.01



(b) Point cloud downsampled using voxels of size 0.01



(c) Voxel Grid of size 0.05

(d) Point cloud downsampled using voxels of size 0.05

Figure 2.2: Voxelization results on point cloud representation of Standford Bunny

This representation is a very well defined data structure and can be used directly with 3D CNN architectures but the voxelization process requires high memory usage

and suffers from information loss which makes it inefficient.

Mesh Representation
A mesh is a surface-based representation method and one of the most commonly used ones in the 3D community. It consists of sets of vertices, edges and faces which all together define the shape and volume of a 3D object/scene. The faces in a mesh can be triangular (triangle mesh), quadrilateral (quad mesh) or even a convex polygon (n-gonal mesh). The advantage with mesh representation is that it does not suffer from information loss like a voxel grid or a point cloud but it is also an unnatural data structure as an input to a neural network.

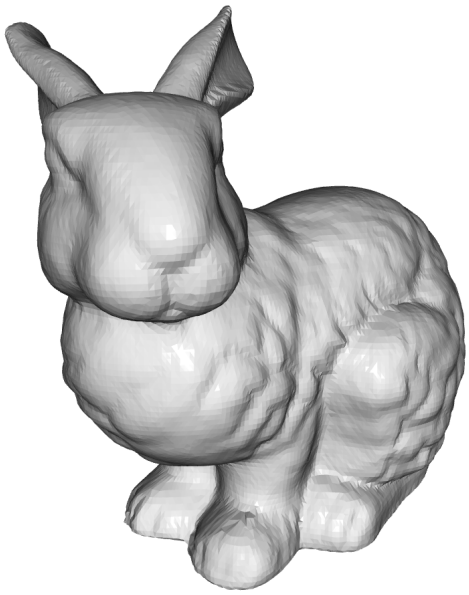
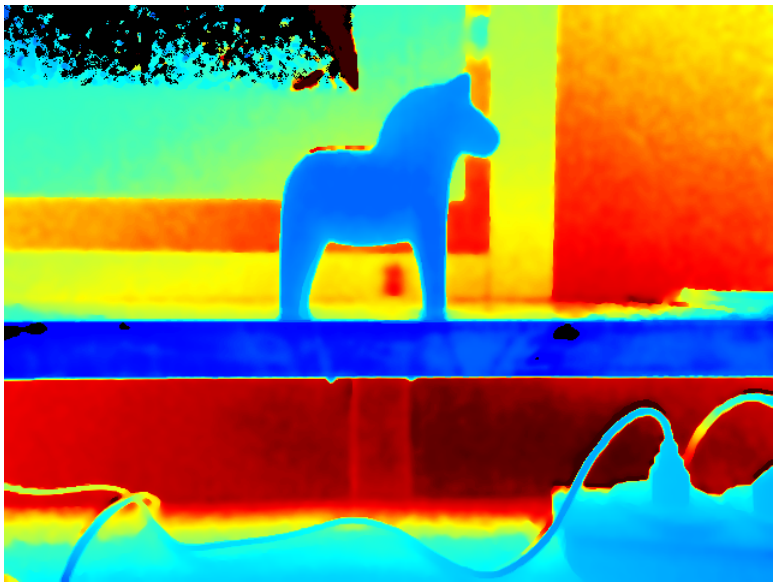


Figure 2.3: Mesh representation of Standford Bunny

Depth Images
Depth images are mainly available as RGBD images i.e. colour images with depth information obtained in the form of distance from the camera origin. This depth information is available for each pixel in the image. However, it is relatively difficult to create a model which can work accurately with only RGBD images as an input. Moreover the data acquisition with respect to the depth information is sensitive to lighting conditions as well as the material of the objects present in the scene.

(a) RGB Color image



(b) Depth image

Figure 2.4: Corresponding color and depth images captured using Intel's Realsense L515 Camera

Signed Distance function
A signed distance function (SDF) is an implicit representation method i.e. it uses some function f to represent a 3D object. Mathematically, SDF f of a given set ω determines the distance (magnitude) of a point β from the boundary of the set, along with the sign indicating whether the point is within the set or not. Hence, the function f encodes the surface (boundary) of a shape. While the SDF is a very interesting representation method, there has been little work done with it with respect to our problem defined in section 1.1. However, it can be an interesting representation to look into for reconstruction of the 3D object using the results of point cloud segmentation.

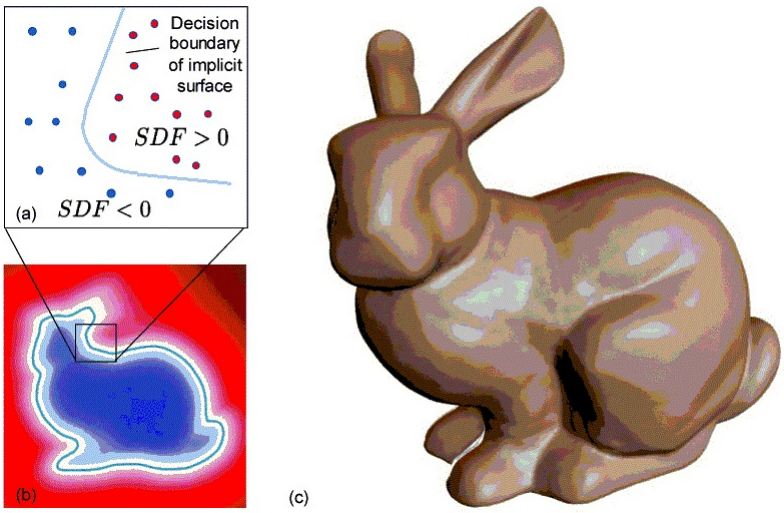


Figure 2.5: SDF learned via DeepSDF[20] applied on Standford Bunny (a) depiction of the decision boundary of underlying implicit surface where SDF < 0 and SDF > 0 means inside and outside of the surface respectively, (b) 2D cross-section of the signed distance function, (c) rendered 3D surface recovered from SDF $= 0$

Point Cloud
A Point cloud is a set of 3D points distributed in a 3D space. Each of these 3D points has a deterministic position denoted by a certain (x, y, z) coordinate and can have more information attributes like RGB colour values. Point cloud representation certainly has more significance over multi-view representation, depth images, voxel grids because they preserve more high-quality geometric information without any discretization. Moreover, they can be sampled directly from a mesh and can also be processed to a voxel grid if at all necessary. However, this representation method loses information about local connections between points which a mesh has. While they may not be the most natural data structure to work on, they are relatively better than most other standard representations. In the next subsection we shall have a more deeper look at point clouds.

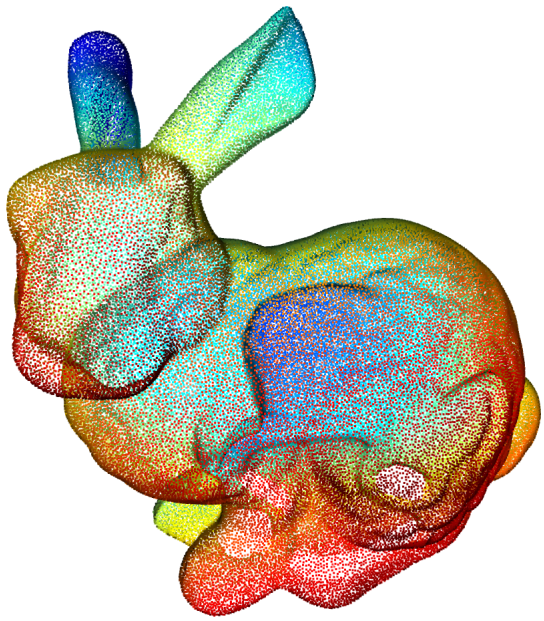


Figure 2.6: Point cloud representation of Stanford Bunny

2.1.1 Point Cloud Theory

A point cloud is a very simple data type, it is basically a set of points in a space. We define a point cloud P in 3D Euclidean space as follows

$$P = \{x_i \, \epsilon \, \mathcal{R}^3\}_{i \leq N} \tag{2.1}$$

where N is the number of points in a given point cloud. Typically, a point cloud would consist of only spatial coordinates $\{x, y, z\}$ but can include more information like colour information in terms of RGB values for each point or normals of each point expressed as (nx, ny, nz). In that case, we can define P as follows

$$P = \{(x_i, f_i) \, | \, x_i \, \epsilon \, \mathcal{R}^3, \; f_i \, \epsilon \, \mathcal{R}^D\}_{i \leq N} \tag{2.2}$$

where f_i represents the D-dimensional additional point feature vector.
Since point clouds are just collections of unordered points they have no information about the connectivity of the points.

2.1.2 Challenges with Point Clouds

When working with point clouds, one has to deal with new challenges which have not been addressed before when working with neural networks. Below, we list some of the challenges encountered when working with point cloud data and their possible solutions.

Varying number of points:
When working with 2D CNNs, the input to such models is well defined structured data. For example, in tasks like image classification, images of varying size are pre-processed before feeding as an input to the CNN model. The pre-processing step can

involve re-sizing all the images to the same size, hence the same sized pixel arrays will represent every image.
In case of point clouds, we may not have the same number of points representing an object. The simplest solution would be to sample a fixed number of points from the point cloud. However, the source of the point cloud as well as the sampling strategy used plays an important role in which points are sampled and eventually shall affect our model. The most commonly used sampling methods are uniform sampling and the farthest point sampling (FPS). The objective of FPS is to sample a fixed number of points Q from a set of N points, such that all the sampled points are farthest from each other. FPS has a certain randomness and non-uniformity associated with it whereas uniform sampling works by creating a 3D voxel grid from the point cloud data and the points closest to the voxel center are then selected.

Irregularity (unorderedness):
Point cloud data is an unordered set which means that the system does not know which point has a higher preference or if all the points have same level of importance. Here, we define the concept of permutation π which is a bijective function of a point cloud, mapping the index set onto itself. Let the original point cloud be defined as follows

$$P = \{x_1, x_2, x_3, \ldots, x_N\} \tag{2.3}$$

By using the permutation

$$\pi : \{1, 2, 3, \ldots, N\} \longrightarrow \{1, 2, 3, \ldots, N\} \tag{2.4}$$

we obtain the reordered point cloud,

$$P_\pi = \{x_{\pi 1}, x_{\pi 2}, x_{\pi 3}, \ldots, x_{\pi N}\} \tag{2.5}$$

If we have a function f such that for all permutations π, we have :

$$\square(x_1, x_2, x_3, \ldots, x_N) = \square(x_{\pi 1}, x_{\pi 2}, x_{\pi 3}, \ldots, x_{\pi N}) \tag{2.6}$$

then such a function $\square$ is called a symmetric function. Examples of such functions are *max*, *min*, *sum*, *average* and *product*. Here, we give a simple example of using *max* (column-wise maximum) applied on a point cloud P and P_π. P consists of 5 points, represented by (x, y, z) coordinates and no additional features like color or normal information and P_π represents a reordered version of P. For ease of viewing, the maximum values in each column are made darker.

$$P = \begin{bmatrix} 0.1 & 0.2 & \mathbf{0.5} \\ 0.5 & 0.1 & 0.2 \\ 0.1 & 0.2 & 0.1 \\ 0.2 & \mathbf{0.5} & 0.4 \\ \mathbf{0.7} & 0.3 & 0.3 \end{bmatrix} P_\pi = \begin{bmatrix} 0.2 & \mathbf{0.5} & 0.4 \\ 0.5 & 0.1 & 0.2 \\ \mathbf{0.7} & 0.3 & 0.3 \\ 0.1 & 0.2 & \mathbf{0.5} \\ 0.1 & 0.2 & 0.1 \end{bmatrix} \tag{2.7a}$$

$$max(P) = max(P_\pi) = (0.7, 0.5, 0.5) \tag{2.7b}$$

The conclusion to draw over here is that one must have some transformation process so that raw point cloud data can be transformed into an input suitable for neural networks or we must design the neural networks so that they are "permutation invariant". To achieve permutation invariance, the use of symmetric functions can be one suitable strategy.

Noise and outliers in point clouds:
Point clouds can be obtained directly from sensor's like LiDAR (Light Detection and Ranging), derived from images or estimated from RGBD maps. Each method of point cloud accquisition results in a point cloud with different point density, different additional feature information. A LiDAR can have a point density less than 10 points/m^2 and even more than 100 points/m^2 depending on the sensor. Point clouds obtained from RGBD images usually have a point density between 10 to 100 points/m^2 whereas for point clouds dervied from images it depends on the spatial resoulution of the cameras used. When point clouds are estimated then the resolution of their depth estimation depends on the underlying model used for point cloud construction. While the LiDAR sensor gives a direct point cloud, its accuracy is dependant on the lighting conditions and the material of surface present in its field of view. Depending on how the point cloud is accquired, there maybe missing points, outliers present and even irregular distortions.

2.2 Deep Learning Primer

Deep learning is a machine learning technique based on the fact that there is enough training data available. Deep learning architectures are based on Artificial Neural Networks (ANN). The concept of artificial neuron develops from the neurons in the human brain, which work by receiving inputs from other neurons and give an output. Mathematically we model artificial neurons (also known as perceptrons) to have an input passing through an activation function which would decide the final action. Figure 2.7 represents a perceptron which takes inputs $\mathbf{x} := x_1, x_2, \ldots, x_N$ and produces an output y. Each input value x_i of $\mathbf{x}$ is associated with a weight value w_i in weight vector $\mathbf{w}$ which indicates the significance of the input value. For the perceptron to produce an output, it needs to be "fired up" just like a real neuron, the condition for this firing is the weighted sum of the inputs must be more than the set threshold of the perceptron. Alternatively one can define the negation of the threshold as the bias b of the preceptron and then incorporate it in the input vector itself by setting $x_0 = b = -\text{threshold}$ and $w_0 = 1$. This pre-activation result is denoted by z and passed as an input to the activation function σ for final output. Hence, we come to the following formulation

$$y = \sigma(z) \tag{2.8}$$

$$\text{where, } z = b + \sum_{i=1}^{n} w_i x_i \tag{2.8a}$$

$$z = \sum_{i=0}^{n} w_i x_i \tag{2.8b}$$

$$z = \mathbf{w}^T \mathbf{x} \tag{2.8c}$$

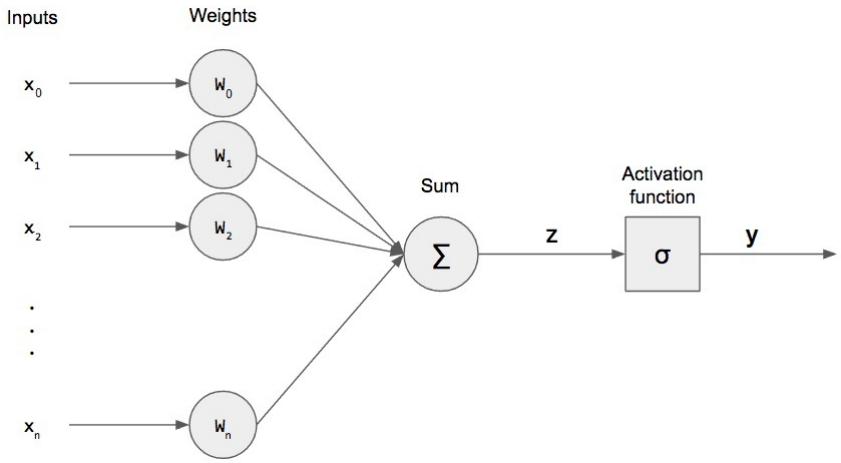


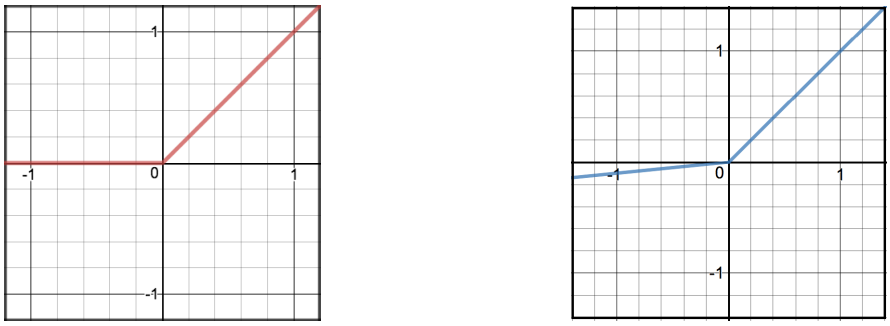
Figure 2.7: A perceptron taking input vector $\mathbf{x}$, multiplying it with weight vector $\mathbf{w}$ and producing output y after processing it via its activation function σ.

2.2.1 Activation Functions

The activation function of a perceptron defines its output. We use Rectified Linear Unit (ReLU) and one of its variants, leaky ReLU in our models. Mathematically ReLU is a *max* function with linear behaviour for positive values and 0 for negative values. Leaky ReLU a is modification of ReLU which allows negative values of the input to exist by scaling them down by a value of α. The functions are defined as follows:-

$$\sigma = \begin{cases} max(0, z) & \text{, if } z \geq 0 \\ \alpha z & \text{, if } z < 0, \text{ ReLU: } \alpha = 0, \text{ Leaky\_ReLU: } \alpha \in (0, 1) \end{cases} \tag{2.9}$$

Figure: 2.8 provides a simple visualization of the respective activation functions.



(a) ReLU **(b)** Leaky ReLU

Figure 2.8: Visualization of Activation Functions: ReLU and Leaky ReLU

2.2.2 Multi-layer perceptron

The perceptron presented by equation 2.8 represents a single layer perceptron. Now we build on this idea to develop the a network with arbitary number of layers containing multiple perceptrons. This type of network is known as a Multilayer perceptron (MLP), it is also known as a "vanilla" neural network or a feedforward neural network. It consists of an input layer to receive the inputs, one output layer that makes a prediction and an arbitary number of hidden layers in between them. Since it is quite understood that every perceptron layer has an input and output, we shall consider the total number of layers in a MLP by only counting the number of hidden layers. Figure 2.9 is an example network graph of a 3-layer perceptron with 4 units in the input layer represented by red boxes, 3 units in the output layer represented by green boxes. The 1^{st}, 2^{nd}, and 3^{rd} hidden layers consist of 4, 3 and 2 perceptron units respectively which are represented as a blue circle. For convenience of visualization, the weights associated with each input are not displayed in the figure.

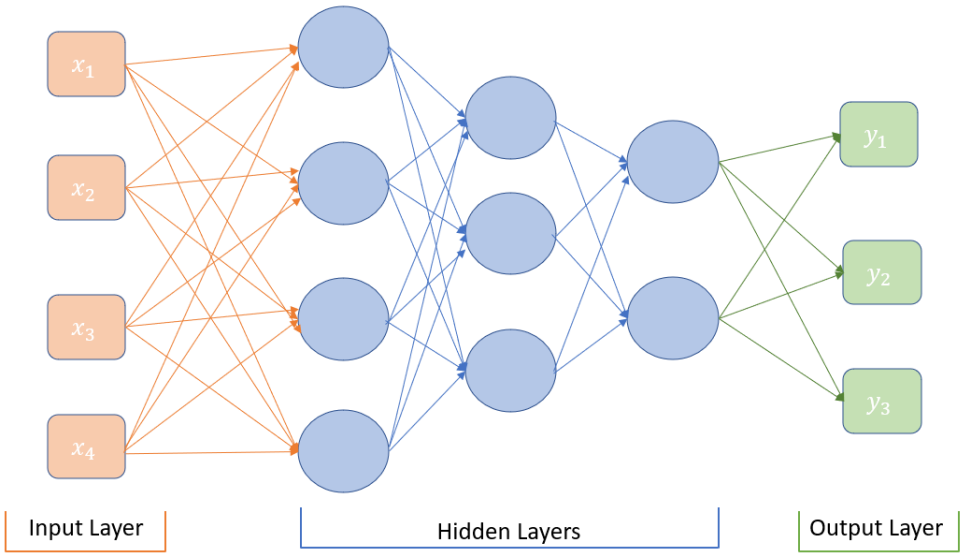


Figure 2.9: Network graph for a 3-layer perceptron

2.2.3 Loss Function and Optimization

Part segmentation is essentially a multilabel (or multiclass) classification problem i.e. the model must identify the label of every point in an input point cloud. An object point cloud can have $m > 0$ parts and for each part a new label is assigned. The loss function that is used for training all the models is called Cross-Entropy Loss. Cross entropy indicates the distance between the predicted output distribution of a model and the original distribution. Following equation 2.10 represents calculation of a loss for each class, where y_c is the prediction made by the neural network for a class c for an input sample s and gt represents the corresponding true values for the same class and input sample.

$$loss = -\sum_{c=1}^{m} gt_{s,c} \log(y_{s,c}) \tag{2.10}$$

Now that we have our loss function, can come to the optimization aspect of our network. For our network to learn and we must tell it to update its weight matrix such that the loss function is minimized. Todo so we use the stochastic gradient descent (SGD) method as it reaches the minima in a shorter amount of time than algorithms like gradient descent and newton's method.

2.2.4 Convolutional Neural Networks

Convolutional neural networks (CNN) are a different type of neural network which have a special structure and are typically used for data which have a grid-like topology. This is because they were designed to find patterns in the data using the convolution operation. There are two main layers in a CNN, the first is the convolution layer and the second is the pooling layer. The convolution layer as the name suggests performs convolution using a convolutional kernel or filter which is slided

accross the input data. The output of convolution is called as a feature map or an activation map. Figure 2.10 shows the output of convoluting a 4×3 with a 2×2 kernel.

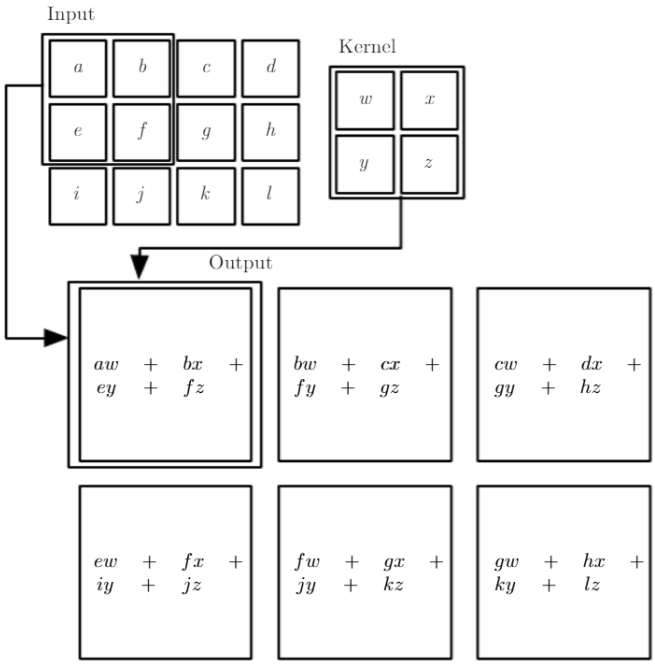


Figure 2.10: Convolution operation

A pooling layer is also known as a subsampling layer which reduces the sample size of a feature map, making the processing faster. The output of a pooling layer is called a pooled feature map. The function used to perform pooling is usually either *max* or *avg*. Following figure 2.11 shows the result of *max* pooling operation on an example feature map.

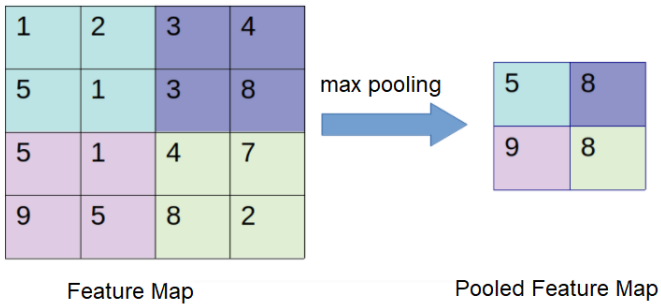


Figure 2.11: Demonstration of max pooling in CNN

A CNN is constructed by stacking multiple convolution and pooling layers. By doing so different layers capture different features present in the input data. Finally a fully connected layer, i.e. MLP, is used to generate a prediction. Figure 2.12 shows a simple CNN architecture consisting of an input layer, combination of convolution

layers and pooling layers acting one after the other, and the result of final pooling layer being sent to the fully connected layers which give us the output/prediction.

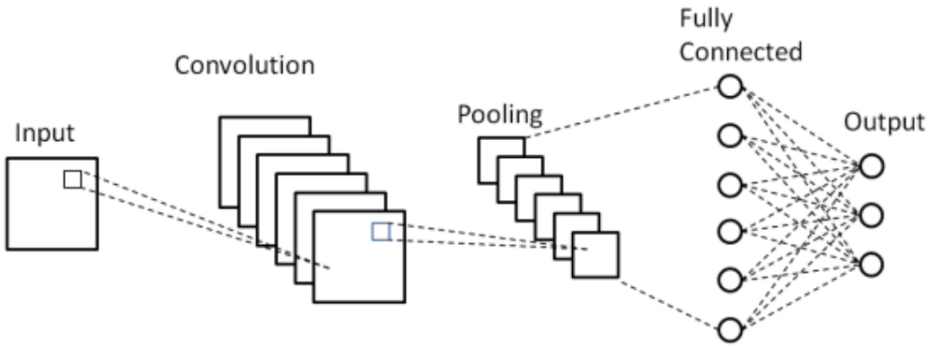


Figure 2.12: High level view of a simple CNN architecture

2.3 Introduction to 3D Deep Learning

2.3.1 Projection Networks

The idea of a projection network is as the name suggests, to project the 3D point cloud data to a more suitable data format. The inspiration for these networks started from one of the earliest and most simplest methods which was to use multiple 2D images which would be generated from 3D point cloud projections; and then process these 2D images with state of the art 2D CNN models as done in [29]. Combining actual multi-view depth maps, 2D images and 3D-to-2D projections come at a computational cost. Similar works on projecting the 3D data to tangents and then performing tangent convolutions like [30] also exists and shows great ability to process large-scale point clouds but they are dependent on the ability of the model to estimate the tangent. Moreover projection based methods are sensitive to viewpoints, cannot exploit underlying geometry and suffer from structural information loss.

A different type of projection network, sometimes referred to as a discretization-based network, which projects the 3D point cloud to sparse lattice structures or volumetric representations have been introduced in [28] and [39] respectively. While the models made for these networks provided good results, volumetric representations require high memory usage for fine resolution and suffer from information loss inherently. Moreover since they are naturally sparse in nature, using dense convolutions was inefficient in the first place. However, projection of 3D point clouds to lattice structures has more advantages. Models like Sparse Lattice Network (SPLATNet) can project both multi-view 2D images and 3D point clouds to high dimensional permutohedral lattice [14] and operate on them with Bilinear Convolutional Layers [28], delivering very good results. However, they are still limited by the grid structure of lattice which eventually puts a constraint on the convolution operation. This thesis does not implement any type of projection or discretization based networks.

2.3.2 Point-wise Networks

Point-wise networks are a new type of neural network which are designed specifically to work with point cloud data. The focus of these networks is to try to learn features directly from the 3D points as an input. Moreover these networks do not have a convolution element to them and therefore, making them completely different from the regular deep learning approaches which are based on convolution. There are two significant models in this area, PointNet [22] and its successor PointNet++ [23], which are explained in sections 3.2 and 3.3 respectively.

2.3.3 Graph-based Convolution Networks

Use of graphs to add structure to unstructured data has been gaining attention in the recent years due to the rise of convolution operators defined specifically for graph-based neural networks. There are two types of convolution operations possible on a graph: spectral and spatial. Spectral-based convolution approaches like [36] carry out multiplication of filters with spectral representation of the graph in the Fourier domain whereas spatial-based convolution methods put their focus on aggregating features from edges for each node. The graph-based approach explored in this thesis is called Dynamic Graph CNN (DGCNN) [34] and is explained in section 3.4.

2.3.4 Point Convolution Networks

Convolution operation can be seen as a template-matching operation when dealing with images. One of the main reasons why CNNs have been so successful in dealing with images is because of the structure of images i.e. images are dense and discrete, and they can be easily represented by arrays while preserving meaning of the pixel intensities. Point clouds on the other hand are sparse and continuous and hence we need a new way to define convolution on point clouds. Point Convolution network formulations are dominated either by the use of MLPs to formulate the kernel or by the use of explicitly defined geometric kernels. A point convolution network explored in this thesis is based on a new convolution operator called Kernel Point Convolution (KPConv) [31] and is explained in section 3.5.

3

Methodology

3.1 Introduction

Selecting the appropriate models to implement from a long list of models was a tough a choice. One motivating factor was to see the use of these models for some real world applications like in [7, 21, 2, 12, 27]. Finally PointNet, PointNet++, Dynamic Graph CNN and Kernel Point Convolution for point clouds were decided to be implemented. The following sections go in depth about each of the networks.

3.2 PointNet

The idea of PointNet is to compute features for every point in the point cloud directly. The input to this model is a $N \times F$ point cloud, where N is the number of points and F is the number of information channels. F is expressed as the $3 + \mathcal{D}$, where 3 represents the 3-dimensional cartesian coordinates and $\mathcal{D}$ represents the additional features. For example, for $\mathcal{D} = 0 \implies F = 3$ we have spatial coordinates (x, y, z) and an input point cloud of size $N \times 3$ or with $\mathcal{D} = 3 \implies F = 6$ we could add the three RGB values per-point, then our input vector would look like this (x, y, z, r, g, b) and an input point cloud of size $N \times 6$. The architecture of PointNet helps it to achieve tasks like object classification, part segmentation and semantic segmentation. For segmentation the output is a prediction matrix of size $N \times m$ i.e. for each of the N points, there is a prediction made for each of the m labels. PointNet uses symmetric functions (refer to Section: 2.6) to bypass the problem of irregularity in point clouds.

$$\begin{aligned} X = \{x_1, x_2, \ldots, x_N\} \subset \mathcal{R}^F,\ N \in \mathcal{R} \\ \text{where, } F = 3 + D \end{aligned} \tag{3.1}$$

Considering the simplest case where the input point cloud has only N number of 3D cartesian coordinates i.e. $D = 0 \implies F = 3$, so we have

$$\begin{aligned} X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \\ \text{where, } x_i = (x, y, z), i \in \{1, 2, \ldots, N\}, N \in \mathcal{R} \end{aligned} \tag{3.2}$$

We can then define the PointNet function f_{pn} as follows

$$\begin{aligned} f_{pn}(X) &= \gamma(\square(\mathbf{h}(X))),\ \mathcal{R}^{N\times F} \longrightarrow \mathcal{R}^k \\ \text{where, } i &\in \{1, 2, \ldots, N\},\ N \in \mathcal{R} \\ \mathbf{h} &: \mathcal{R}^{N\times F} \longrightarrow \mathcal{R}^{N\times M},\ M \in \mathcal{R} \\ \square &: \mathcal{R}^{N\times M} \longrightarrow \mathcal{R}^{1\times M} \\ \gamma &: \mathcal{R}^{1\times M} \longrightarrow \mathcal{R}^k \end{aligned} \tag{3.3}$$

Here the function $\mathbf{h}$ indicates a list of MLP's, γ is implemented as a MLP and $\square$ represents a symmetric function. The proposed PointNet model in [22] uses $M = 1024$ which means that $\mathbf{h}$ consinsts of 1024 MLPs, each MLP takes in a point x_i as input and ouputs a scalar value. So the function $\mathbf{h}$ can be modelled as follows

$$\begin{aligned} \mathbf{h}(X) &= (h_1(x_i), h_2(x_i), \ldots, h_{1024}(x_i)),\ \mathcal{R}^{N\times 3} \longrightarrow \mathcal{R}^{N\times 1024} \\ \text{where, } h_j &: \mathcal{R}^3 \longrightarrow \mathcal{R} \\ j &\in \{1, 2, \ldots, M\},\ M \in \mathcal{R} \end{aligned} \tag{3.4}$$

The function h would be applied on all N points resulting in a $N \times 1024$ matrix. These extracted features (per-point features) are then aggregated using the symmetric function $\square = max$. This results in a global feature vector of size 1×1024. Finally, γ represents a new MLP which processes the global feature vector to output a point set feature of size k.

To address the problem of invariance to rigid transforms, PointNet introduced a spatial transformer network (STN) which aligns an arbitrary point cloud to a canonical space. This network is sometimes also called as the Joint Alignment Network and allows spatial manipulation of the data within the network itself. The idea of this network is to somehow canonize the input point cloud. STN is used twice in the PointNet model, once as an input transfer where the output is a 3x3 matrix and second time as a feature transformer where the output is a 64x64 matrix.

The PointNet model architecture [22] consists of two networks. The first network is called the Classification Network whose core base is the PointNet function mentioned in Section 3.3. The second network is called the Segmentation Network whose task is to output per-point scores for m semantic subcategories. The input to this network is the global feature vector k as well as the per-point feature vector which is the result of $\mathbf{h}$ function. This network acts as input transformer as well as a feature transformer when the input is being processed in the Classification Network. An overview of the PointNet architecture can be seen in Figure: 3.1. The complete network architecture details can be found in [22].

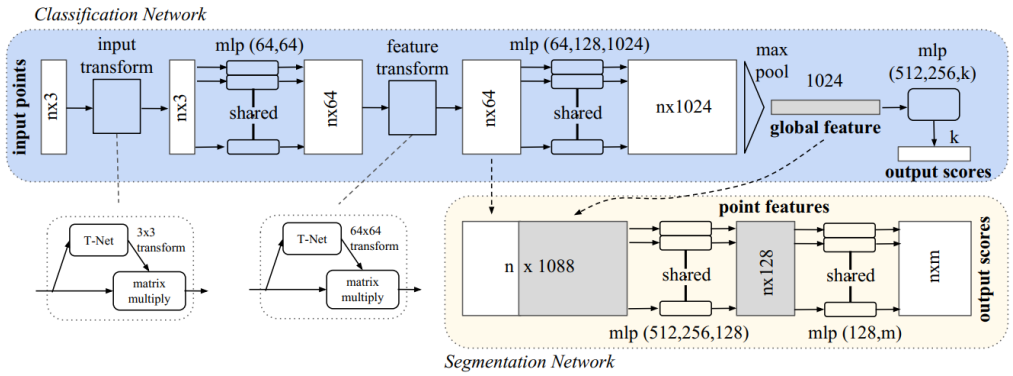


Figure 3.1: Network Architecture of PointNet

3.3 PointNet++

PointNet++ is the successor to PointNet architecture. The reason to build this model was to overcome the main drawback of PointNet, namely that PointNet did not really capture interaction between points. To capture this point interaction, PointNet++ introduced a new layer called Sampling and Grouping Layer. The result of this layer is then fed into PointNet Layer for feature extraction. The combined process of Sampling and Grouping Layer followed by a layer PointNet is called Set Abstraction Layer (SAL). PointNet++ is based on multiple SAL which helps it to achieve local feature aggregation for every point. We know introduce the Sampling and Grouping Layer.

Sampling Layer

1. This layer is used to generate centroids from the given input point cloud. The centroids are sampled using the Farthest Point Sampling (FPS) algorithm.
2. The objective of this algorithm is to sample a fixed number of points Q from an input point cloud P and output a sampled point cloud P_q such that all the points in P_q are farthest from every other point in P_q for the same value of Q. This methods provides a good coverage of the entire point cloud. For an algorithmic implementation please refer to A.1.

$$P = \{x_1, x_2, \ldots, x_N\} \tag{3.5a}$$

$$P_q = \{x_1, x_2, \ldots, x_Q\} \text{ where } Q < N \tag{3.5b}$$

Grouping Layer

1. The objective of this layer is to find a fixed number of nearest neighbours for all of the points in P_q. There are two possible approaches to do this: by using the ball query search or the k-Nearest Neighbours (kNN) algorithm. kNN is a simple algorithm which shall find a fixed number of neighbours from every sampled point based on a distance metric. Wheras ball query search forms a sphere/ball with every point in P_q and a predefined radius, and then returns all the points within the sphere. For an algorithmic implementation please refer to A.2.
2. The input to this layer is the original input point cloud P and the sampled

point cloud P_q, the output of this layer is a neighbourhood point set $\mathcal{N}_m \, \forall \, q \in Q$ in the Euclidean space.

An overview of this architecture can be seen in Figure: 3.2. The complete network architecture details can be found in [23].

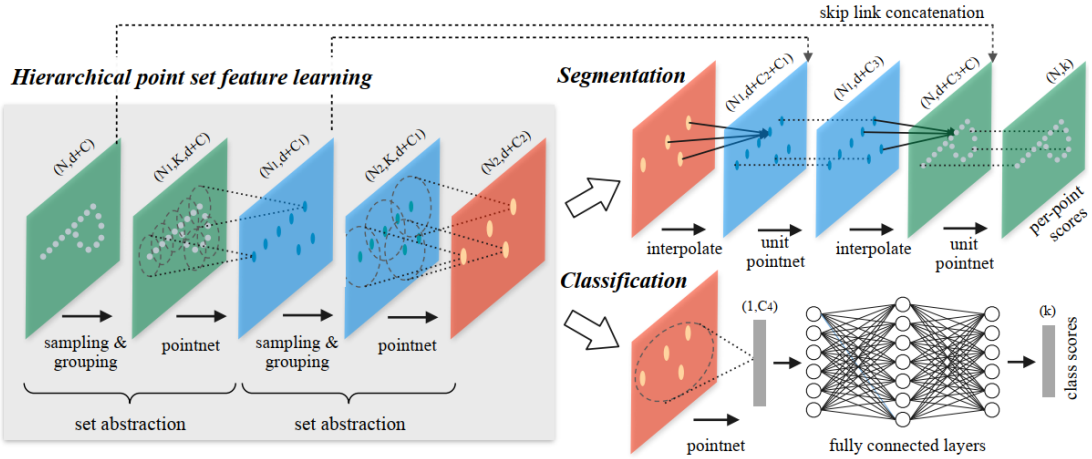


Figure 3.2: Network Architecture of PointNet++

3.4 Dynamic Graph CNN

Dynamic Graph CNN is a graph-based approach to solve classificaiton and segmentation problems of point cloud data. It creates a graph from the point cloud data by finding a fixed number of nearest neighbours of the input data points and then proceeds to compute features and apply convolution. The original paper [34] introduces the concept of "EdgeConv" operator which eventually replaces the MLP used in the PointNet. The model takes points, can take additional features as an input, it then computes a neighbourhood point set $\mathcal{N}_i$ for every point and then calculates edge values using "EdgeConv" operator for each point in every $\mathcal{N}_i$; then it aggregates the results for every $\mathcal{N}_i$. It performs very well for segmentation, but the computational complexity of the model increases very fast as it is a graph-based method in which the graph is updated after every layer. Figure:3.4 shows the architecture of the model for classification and segmentation. Following is the explanation of the EdgeConv and layer update operation.

Edge Convolutions (EdgeConv)

1. We start by using the simplified equation mentioned in Section 3.6.

$$X = \{x_1, x_2, \ldots, x_N\} \subset \mathcal{R}^F, \; N \in \mathcal{R}$$
$$\text{where, } F = 3 + D$$

2. Compute a directed graph G using vertices, V and edges, E. Vertices of this graph are from X i.e. the points themselves and the edges which represent the connectivity between different vertices are computed using k-nearest neighbours algorithm. The graph also includes a "self-loop" which means that every

vertex is also connected to itself. The graph $G = (V, E)$ now represents a local point cloud structure where

$$\begin{aligned} V &= \{1, 2, \ldots, N\} \\ E &\subset V \times V \end{aligned} \tag{3.6}$$

3. Next we compute the edge feature as follows

$$\begin{aligned} e_{ij} &= h_\Phi(x_i, x_j) \\ \text{where, } h_\Phi &: \mathcal{R}^F \times \mathcal{R}^F \rightarrow \mathcal{R}^{Fn} \\ i &\in V \\ j &\in V \end{aligned} \tag{3.7}$$

Here h_Φ is a nonlinear function with learnable parameters Φ

4. Convolution and feature aggregation! To convolute point x_i we make use of the neighbourhood information of the point x_i given by $j : (i, j) \in E$, the edge features e_{ij}, and a symmetric function (explained here 2.6) denoted by $\square$. The output $x_i^{'}$ is then given by

$$\begin{aligned} x_i^{'} &= \square_{j:(i,j) \in E} \; e_{ij} \\ \text{where, } e_{ij} &= h_\Phi(x_i, x_j) \end{aligned} \tag{3.8}$$

Figure:3.3 gives a visual representation of the EdgeConv operator.

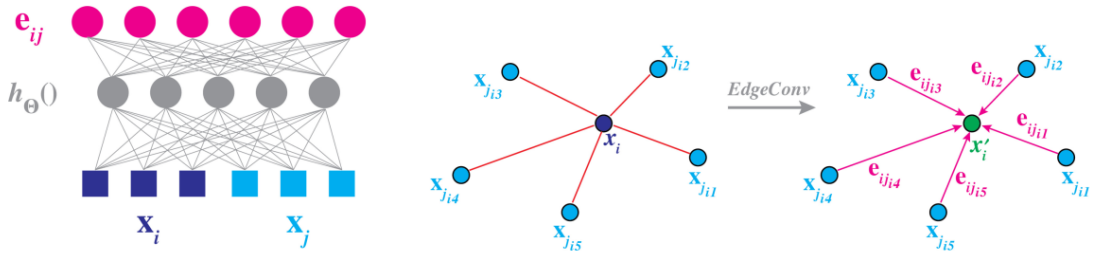


Figure 3.3: Visual representation of EdgeConv Operation of DGCNN

Selecting correct $\square$ and $h_\Phi(x_i, x_j)$ functions is important here as they have crucial influence on the properties of EdgeConv. For $\square$ its adviseable to use *max* function. $h_\Phi(x_i, x_j)$ is implemented as a MLP but the input to MLP can vary. For example, if we consider PointNet then $h_\Phi(x_i, x_j) = h_\Phi(x_i)$, hence the MLP extracts features based on only the point x_i. The original paper [34] suggests 3-4 options for selecting $\square$ and h_Φ.

Update Layer

1. For every layer l the graph G^l is recomputed using the k-nearest neighbours approach.
2. The nearest neighbours are computed in the feature space using L_2 distance.
3. Using a symmetric function like max-pooling, permutation invariance is by-passed.

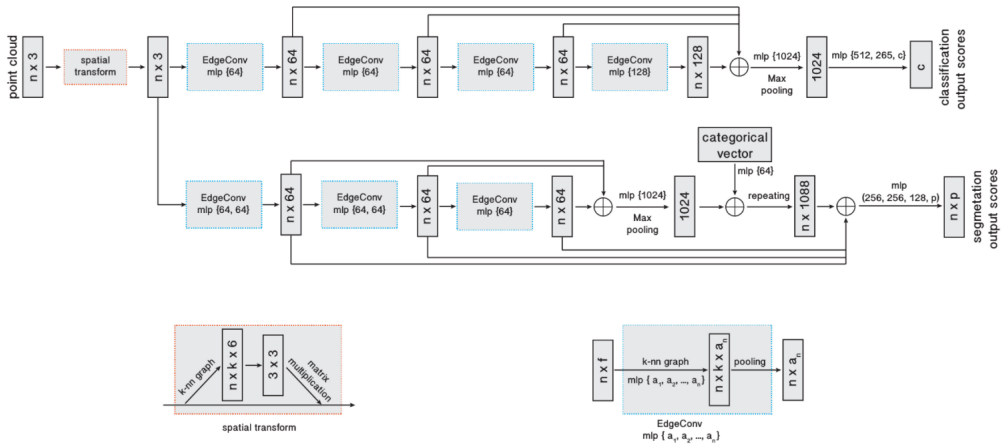


Figure 3.4: Network Architecture of Dynamic Graph CNN

3.5 Kernel Point Convolutions

Kernel Point Convolutions (KPConv) for point clouds [31] proposed a new type point convolution method which was meant to overcome the limitations of previous architectures. To achieve the, author proposed a kernel function to compute point-wise filters. In convolutions on 2D images, we have the kernel weights associated with pixel values and the filters we use are discrete and fixed in size. In KPConv, we have kernel points instead of pixels, and they are used to determine the kernel weights. We then use this kernel function to iterate over small neighborhoods of the point clouds, this is analogous to using 2D kernel filters and traversing them accros an image. To understand the formulation of convolutions in KPConv we begin by redefining original point cloud equation (defined in equation2.2) for the scope of this section. We split our P into two parts, in the following way

$$\mathcal{P}_{KP} \in \mathcal{R}^{N \times 3} \tag{3.9a}$$

$$\mathcal{F}_{KP} \in \mathcal{R}^{N \times D} \tag{3.9b}$$

Application of kernel function g_{KP} to input data point $\mathcal{F}_{KP}$ at point x is equal to the weighted sum of neighbouring point features with the neighbourhood $\mathcal{N}_x$ defined as a ball in 3D space with radius $r \in \mathcal{R}$. The convolution operation if defined as follows

$$\begin{aligned} (\mathcal{F}_{KP} * g_{KP})(x) &= \sum_{\hat{x_i} \in \mathcal{N}_x} g_{KP}(\hat{x_i} - x) f_i \\ \text{where, } \mathcal{N}_x &= \left\{ x_i \in \mathcal{P}_{KP} \,\middle|\, ||\hat{x_i} - x|| \leq r \right\} \end{aligned} \tag{3.10}$$

The idea of kernel function g_{KP} is that it must be able to apply different weights to different areas inside a neighborhood. Drawing a comparison to the 2D CNN, where kernels are usually defined as a square matrix of size smaller than input data matrix, KPConv defines the kernel g_{KP} using $\mathcal{K}$ support points which are known as

kernel points. Figure 3.5 provides a comparison between a kernel of 2D CNN and KPConv.

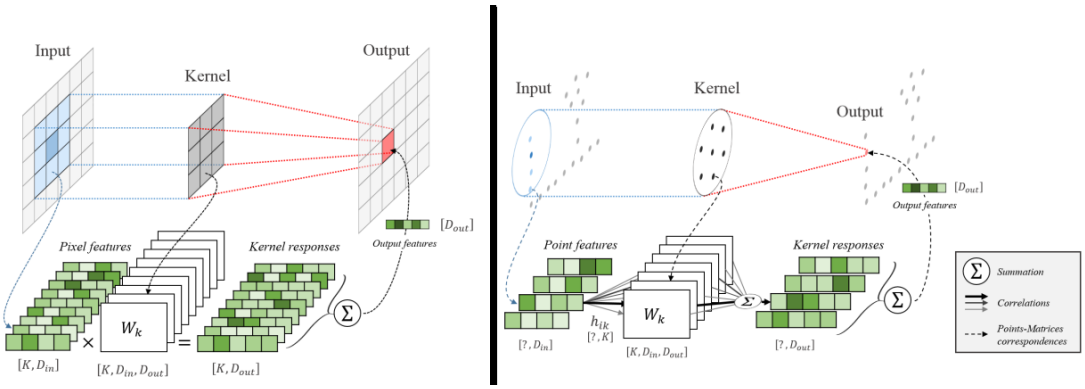


Figure 3.5: Visualization Comparison:
Left: Regular Convolution on 2D Image and Right: Kernel Point Convolutions on Un-ordered 2D Points

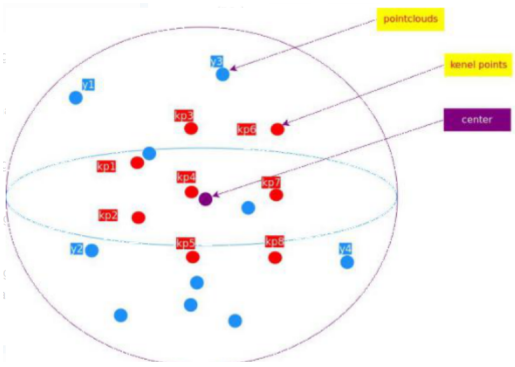


Figure 3.6: Visualization of kernel points, center and points of a point cloud in a neighborhood

The input to this function is neighboring points centered on x and defined as $\hat{x}_i = x_i - x$, the domain of the function becomes $\mathcal{B}_r^3 = \left\{\hat{x}_i \in \mathcal{R}^3 \,\middle|\, ||\,\hat{x}_i \leq r\,||\right\}$. The kernel points are defined by $\{\tilde{x_k} \,|\, k < \mathcal{K}\} \subset \mathcal{B}_r^3$ and have a learnable weight matrix $W_k \,|\, k < \mathcal{K}\} \subset \mathcal{R}^{D_{in} \times D_{out}}$ associated with them. Here D_{in} and D_{out} represent the number of input dimensions incoming from the previous layer and the number of output dimensions. For visual representation of this setting see figure 3.6. The kernel function g_{KP} for a point $\hat{x}_i \in \mathcal{B}_r^3$ is then defined as follows

$$g_{KP}(\hat{x}_i) = \sum_{k<\mathcal{K}} \psi(\hat{x}_i, \tilde{x_k}) W_k \tag{3.11}$$

where, ψ is a linear correlation function which measures how much the kernel point $\tilde{x_k}$ is correlated with each neighboring point $\hat{x}_i$. By logic, the correlation should be

higher for two points which are close to each other and is defined as follows

$$\psi(\hat{x_i}, \tilde{x_k}) = max\left(0, 1 - \frac{||\hat{x_i} - \tilde{x_k}||}{\sigma}\right) \tag{3.12}$$

where, σ is used as an influencer for distance and is selected based on the point cloud density. Kernel point positions are critical to the convolution operator and KPConv provides way to express deformable kernels. However since the results presented in the paper for part segmentation of objects didn't have any substantial difference in them, I worked only with the rigid version of the kernel.
Core Ideas:

1. Two architectures are provided in the original paper called as KP-CNN and KP-FCNN. The first one deals with classification while the later deals with segmentation task. KP-FCNN has the same encoder module with variations in the decoder stage for segmentation. Following we first introduce the network parameters and then the are the basic layers used in both architectures.

2. We first set the cell size dl_l for every layer l. Using the cell size other parameters will be computed. The distance parameter in kernel points is set as $\sigma_l = \Sigma \times dl_l$. We make use of the rigid kernel only for which the convolution radius is set to $2.0 * \sigma_l$. The number of kernel points $K = 15$, $\Sigma = 1.0$ and he first cell size dl_0 depends on the dataset used and in case of ShapeNetPart will be 0.02 i.e. 2 cm.

3. Sampling Layer: Downsampling the point cloud using uniform sampling to reduce the density of inputs at each layer.

4. Pooling Layer: In KPConv, the number of points is reduced progressively, so the cell size is doubled at every pooling layer to increase the effective reception. Finally *max* pooling is used to obtain the pooled features.

5. KPConv Layer: This is layer is the main convolution layer which takes in an input $\mathcal{P}_{KP}$, their corresponding features $\mathcal{F}_{KP}$ and the matrix of neighborhood indices $\mathcal{N}_{mat} \in [[1, N]]^{N' \times n_{max}}$. Here N' is the number of locations where the neighborhoods are computed. The neighborhood matrix is forced to have the size of the biggest neighborhood n_{max}, causing the matrix to have unwanted entries. For points whose neighbours are less than n_{max}, the unwanted matrix entries are simply ignored. Figure 3.7 shows the complete network architectures, [31] contains complete details regarding training the network.

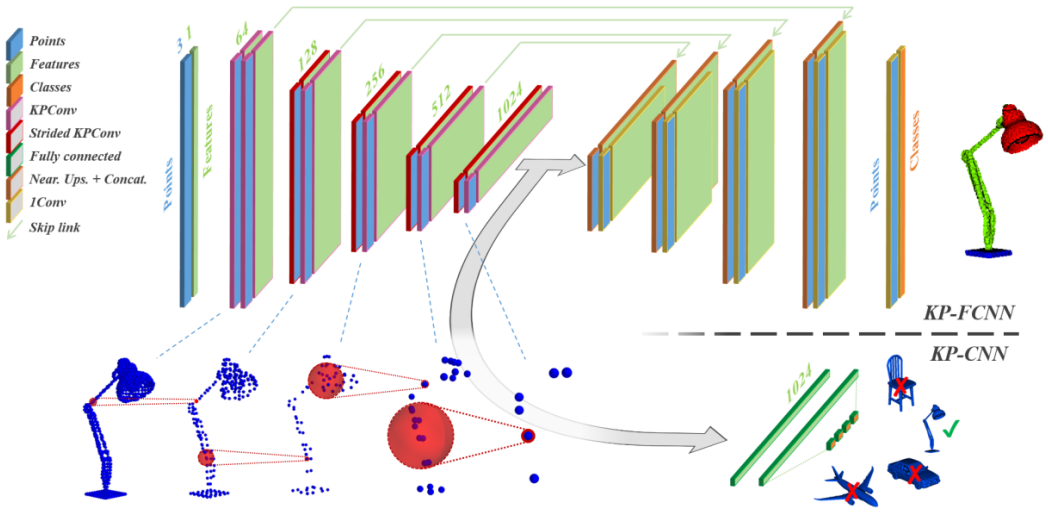


Figure 3.7: Network Architecture of KPConv
KP-FCNN: Kernel Point - Fully Connected Neural Network used for Segmentation

3.6 Evaluation Metrics

To measure the results of segmentation the following metrics were used:

1. Accuracy (Acc) score is expressed as a percentage and is simply calculated as the number of correctly predicted labels over total number of samples i.e.

$$\text{Accuracy} = \frac{100}{N} \sum_{i=0}^{N-1} \mathbf{1}(gt_i = y_i)\,\% \tag{3.13}$$

   where N is the total number of samples, gt_i and y_i is the true and predicted label of the i^{th} sample, respectively.

2. Average Accuracy (Avg Acc) score is useful when the classes are imbalanced. To calculte this metric we need to know sensitvity and specifity of our classifier. Sensitivity is defined as the true positive rate and specificity is defined as the true negative rate. Then the average accuracy is defined as the arithmetic mean of sensitvity and specifity. An outcome is true positive when the model correctly predicts the positive class and it is a true negative when the model predicts the negative class corretly. If the model predicts the positive class incorrectly then the outcome is false positive, simarly if it predicts the negative class incorretly then the outcome of the model is false negative.

$$\text{Sensitivity} = \frac{\textit{true positives}}{\textit{true positives} + \textit{false negatives}}$$

$$\text{Specifity} = \frac{\textit{true negatives}}{\textit{false positives} + \textit{true negatives}}$$

$$\text{Average Accuracy} = \frac{\text{Sensitivity} + \text{Specifity}}{2}$$

where N is the total number of samples, y_i and $\hat{y}_i$ is the true and predicted labels of the i^{th} sample, respectively.

3. Intersection over Union (IoU) captures the ratio of the area of the overlap and the area of the union. Mathematically, for two sets A and B, IoU is defined as follows:-

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.15}$$

Now that we know what is IoU let us define it for our case. Let us say there are S number of shapes/samples from C number of object categories. Each object will have different number of parts, and it is not necessary that the samples belonging to the same object category have same number of parts. Once a prediction is made by the model, we begin by calculating the IoU of each part type in the category C, this is done by computing the IoU between the groundtruth and the prediction. If the union of groundtruth and prediction points is 0, then we count the part IoU as 1. We then average IoUs for all part types in category C to get mIoU for that particular shape. To calculate mean(IoU) for the category, we take average of mean(IoUs) for all shapes in that category. For ease, henceforth the use IoU means this particular result, unless specified as something else.

4
Development

4.1 Hardware

4.1.1 Data Capture Hardware

For data collection, Intel's RealSense LiDAR Camera L515 was used. The L515 uses a Micro electro-mechanical systems (MEMS) to scan an Infrared (IR) sensor over the device's entire Field-Of-View (FOV), the result of which is then processed by an Application-Specific Integrated Circuit (ASIC) to output a 3D point cloud. Table 4.1 below lists the two resolutions available in which one can capture a point cloud, followed by Table 4.2 mentioning the different important specifications of the device. For a complete set of the device's operating modes refer to table: C

Sr. No.	Depth Resolution (w x h)	Number of depth points per second	FOV
1	640 x 480	9.2 million	70^o x 55^o $\pm 2^o$
2	1024 x 768	23.6 million	70^o x 55^o $\pm 2^o$

Table 4.1: L515 operating models

Depth frame rate	30 frames per second
RGB frame rate	30 frames per second
F Number	2.0
Focal Length	1.88mm
Focus type	Fixed
RGB sensor FOV (H × V)	$69^o \times 42^o$ ($\pm 1\ ^o$)

Table 4.2: L515 device specifications

L515 is also equipped with a 6 Degrees-Of-Freedom (DOF) Inertial Measurement Unit (IMU). Raw accelerator and gyroscope values can be obtained from the device. Having an IMU equipped with a camera (and LiDAR) is helpful as one can compute the pose of the sensor using the IMU information and therefore localize the movement of the sensor. By combining this localization information and visual data captured, one can attempt to solve the problem of simultaneous localization and mapping (SLAM). The challenge of SLAM is about moving a device in an environment and developing or updating a map made out of captured visual data (images or point clouds or both) while keeping track of the current position of the sensor.

There are many approaches to solve this task but considering the time constraints and a new study to be performed to achieve good quality results for SLAM, making use of the IMU was not considered for this thesis.

4.1.2 Computation System

The Vera cluster is built on Intel Xeon Gold 6130 CPU's. The system consists of in total 245 compute nodes (total of 7848 cores) with a total of 28 TiB of RAM and 13 GPU's (8 NVIDIA Tesla V100 and 5 NVIDIA Tesla T4). The GPU used for training the models is NVIDIA Tesla V100 32 GB. For exact system bifurcation and GPU details, we refer to [32] and [6] respectively.

4.2 Dataset & Preprocessing

4.2.1 Synthetic Dataset

For training the models, I have used an information rich 3D model repository called ShapeNet [4]. It contains over 3 million shapes from online 3D model repositories. Moreover there are different types of annotations available for the models like part correspondences, part hierarchies, functional annotations of parts, physical annotations like texture and weight. ShapeNetPart dataset contains 16,881 pre-aligned shapes from 16 categories, annotated with 50 segmentation parts in total. The objects in each category are labeled with 2 to 5 segmentation parts. The authors provide the dataset in a compressed format with 12,137 (70%) shapes for training, 1,870 (10%) shapes for validation, and 2,874 (20%) shapes for testing. For the purpose of this thesis ShapeNetPart is an ideal candidate to work with. Before given as an input to the models, the dataset is pre-processed such that the number of points for each object category is sampled to 2048 using uniform sampling and then each point cloud is subjected to random transformations (rotations, translations, jittering).

4.2.2 Test Data Collection

Accurately capturing a 360^o degree point cloud of an object and annotating it is a laborious task. Here I present the method I followed to have a well defined point cloud of my test object: 'mug'. I used two different mugs, the first one is brown in color and is referred as object 1, while the second is white in color and is referred to as object 2. To accomplish this task, I used Intel's RealSense LiDAR L515 details of which can be found in Section 4.1, 3D data processing library Open3D [38] and 3D annotation tool called labelCloud [26]. The data collection pipeline is based on [5] and is explained in more depth in Appendix:B. Below is the summary of steps followed to capture the data using L515 and to pre-process it.

1. Keep the camera stationary and save the camera parameters. Place the object on a box, approximately between 35 cm and 50 cm from the camera origin.

Figure 4.1f shows the image of object 1 on the box which I used for my data collection.

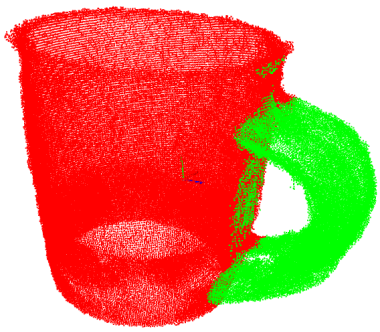
2. Capture RGBD images and point cloud, while manually rotating the object kept on a box. A Graphical User Interface (GUI) was developed to accomplish this task effectively. The result of this is n_pcd point clouds and n_rgbd color as well as depth images.
3. For all the n_pcd point clouds:-
   (a) Compute the distance of each point to the origin $(0, 0, 0)$ and discard the point if its distance is more than 50 cm.
   (b) Using random sample consensus (RANSAC) method we can identify the points which belong in a plane in this above filtered point cloud. The flat planar surface of the box on which the object is placed can be detected to a workable accuracy. Once the plane is identified, we can discard the points which belong to this plane.
   (c) Since one needs to set the parameters for the above RANSAC-based plane segmentation. Using an iterative strategy, one can manually keep tuning different parameters for the first input point cloud till the desired result is obtained. Once optimum parameters are found, they are saved and used to process all the remaining n_pcd-1 point clouds.
   (d) Each of these segmented point clouds are now downsampled using voxel-based downsampling and normals for every point are estimated if not already present.
4. Multiway point cloud registration:-
   (a) Compute registrations for each point_cloud with all point_clouds (n_pcds $*$ n_pcds iterations) $\longrightarrow$ fused_point_cloud. To compute the registrations a color-based iterative closest point algorithm is used, the result of which is a transformation matrix needed for aligning 2 point clouds. Then result of step for object 1 can be seen in figures 4.1a and 4.1b.
   (b) If required use RANSAC-based plane segmentation to remove any existing unwanted plannar surfaces. Use the dimensions of the box as a means of maximum bound, crop the point cloud to get rid of points too far way from the main object. For outliers close to the object use statistical outlier filter. Refer to figures 4.1c and 4.1d for results on object 1. These plane and point removal steps should be performed as required, the objective is to have a good point cloud representation of the real object.
5. Point Cloud Annotation
   (a) Use labelCloud to draw and align 3D bounding boxes for each part of the point cloud. Save the dimensionsions and centroid of these bounding boxes.
   (b) Create an empty vector called label_vector whose length is the same as the number of points in fused_point_cloud and initialize it to 0. Using the geometry information of bounding boxes, find the points which lie within the boxes and set a label value in label_vector exactly at those index locations. Repeat this process for all bounding boxes. The result of labelling object 1 can be seen in figure 4.1e.

(a) Multiway registration result: view 1

(b) Multiway registration result: view 2

(c) Visualization after removing plane using RANSAC-based plane segmentation & mild statistical outlier filter

(d) Visualization after statistical outlier filter with stronger values



(e) Visualization of labels after manually annotating the point cloud

(f) True image of mug with the box on which it kept during data capture

Figure 4.1: Visual results of test data collection process on object 1

5
Results

Following are the numerical and visual results from training and testing the previously explained models. Since it is not possible to have a 3D view in the report, I have presented the visuals in a constrained manner. Typically all the visual images provided here are screenshots of 3D plots. There are two types of views (all views are orthogonal in nature unless specified otherwise) provided: Untransformed view and transformed view. Untransformed views mean that the screenshot of the 3D view was taken without rotating or translating the object, whereas transformed view typically means that the object was rotated and translated for easy comparison. This was necessary because considering an object which was translated quite far from the origin (0,0,0), taking an untransformed screenshot would lead to consumption of too much space. In visualizations, a small coordinate frame also exists. It is represented as 3 different colored arrows, each one of them pointing in a certain direction. The red arrow cooresponds to X-axis, green arrow with Y-axis and finally blue arrow with Z-axis. Ideally all views are arranged such that the viewer see's the Y-axis pointing upwards.

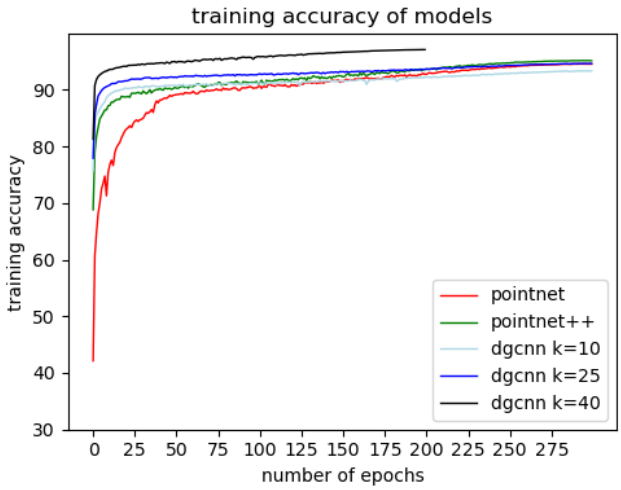
5.1 Results on Synthetic Dataset

The test dataset consists of 2874 samples made of 13 object categories. Since there are so many samples, visualizations of only 1 sample belonging to 2 objects (chair and airplane) are presented. Two tests were performed on the entire test dataset called Test 1 and Test 2. For both the test the number of points was set to 2048 and Test 2 was subjected to random translations, rotations and jittering. A new random transformation was applied for every object, before testing any model.

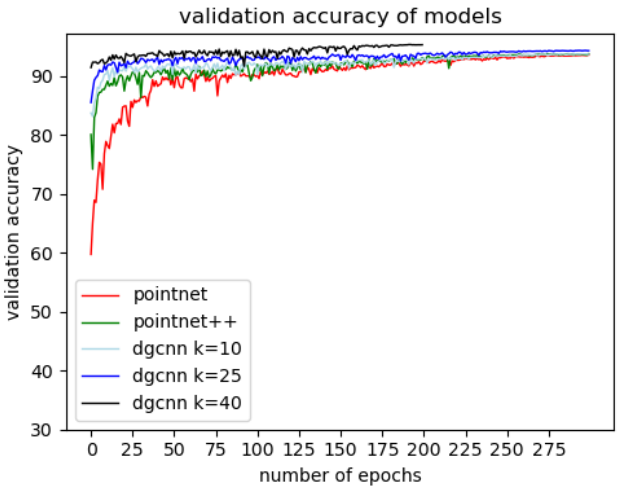
The radius setting of PointNet++ was set to 0.4 and 0.2 in the first and second Set Abstraction Layer respectively. For DGCNN, three models were trained, each of them having a value 10, 25 or 40 as nearest neighbours for every point. The models were trained for a maximum of 300 epochs (iterations). An early stopping mechanism was utilized to stop the training process earlier if the relative increase in validation loss was substantial. For KPConv, the first cell size was set to 0.02, the number of kernel points were 15 and no early stopping was utilized.

For models: PointNet, PointNet++ and DGCNN, one can find the accuracy, loss and IoU for the training in figures 5.1a, 5.1c, 5.1e and the validation in figures 5.1b, 5.1d and 5.1f respectively. For KPConv only training loss and training IoU can be

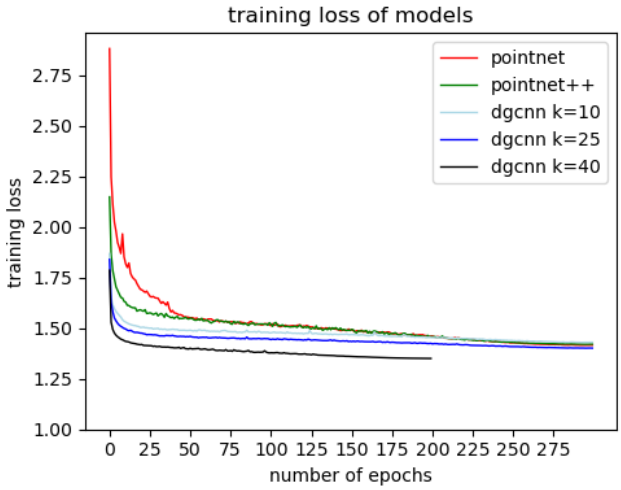
seen in figures 5.1g and 5.1h respectively.



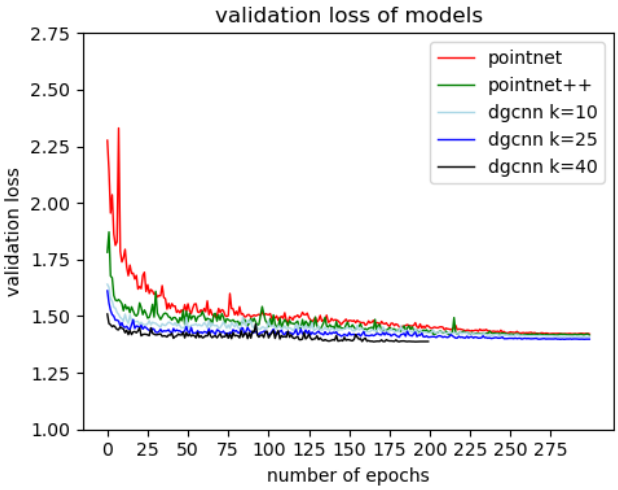
(a) Accuracy of models during training



(b) Accuracy of models during validation stage



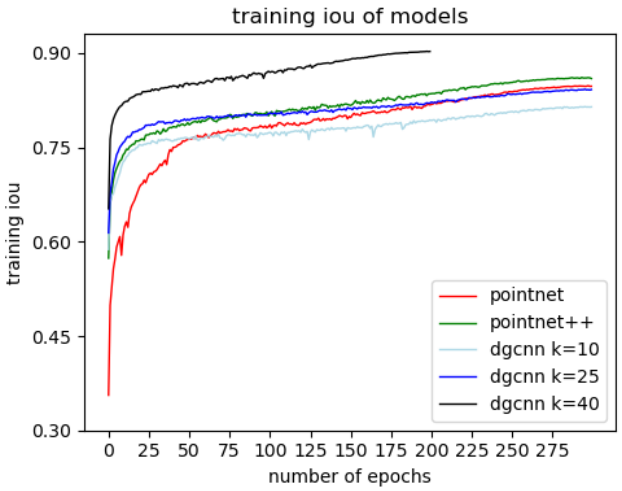
(c) Loss of models during training stage



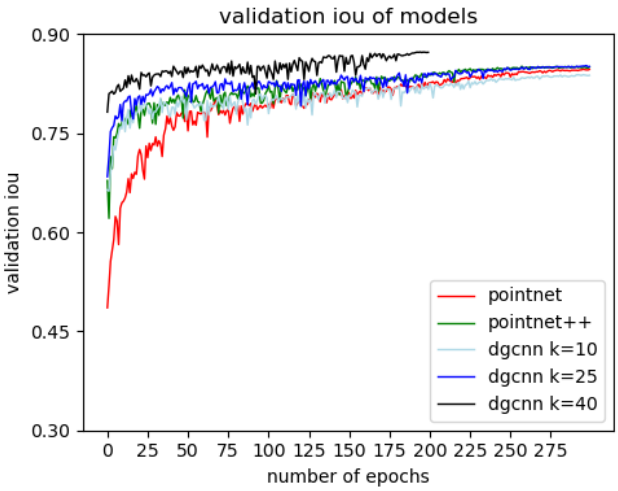
(d) Loss of models during validation stage

Figure 5.1: Training results on ShapeNetPart: training dataset

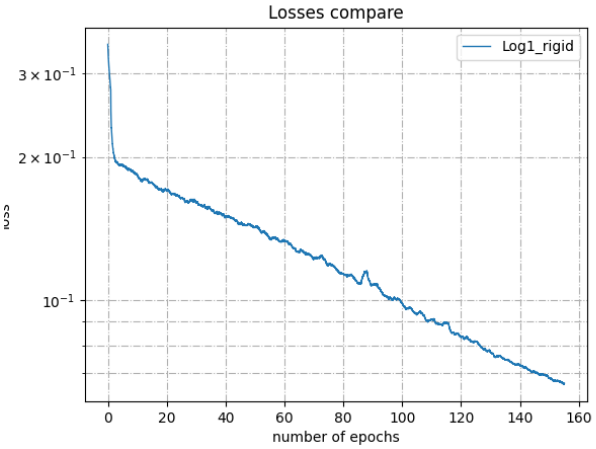
The following table 5.1 provides the training duration for each of the models. This duration does not include the processing time needed to sample the point clouds. The average processing time for the ShapeNetPart dataset is 1 to 3 hours based on the sampling method used.



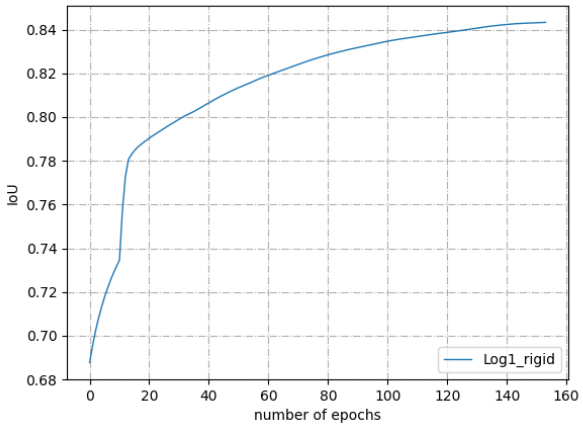
(e) IoU of models during training stage



(f) IoU of models during validation stage



(g) Loss of KPConv during training stage



(h) IoU of KPConv during training stage

Figure 5.1: Training results on ShapeNetPart: training dataset (cont.)

Model	Training Duration
PointNet	0 hr 33 min 3 sec
PointNet++	6 hr 18 min 21 sec
DGCNN (k = 10)	6 hr 31 min 16 sec
DGCNN (k = 25)	9 hr 3 min 10 sec
DGCNN (k = 40)	3 hr 40 min 39 sec
KPConv	72 hours 0 min 0 sec

Table 5.1: Training Duration of all models on ShapeNetPart training dataset

The following table 5.2 provides the three evaluation metrics: accuracy, average accuracy and IoU, and the testing duration on the test dataset. The results are provided for Test 1 and Test2.

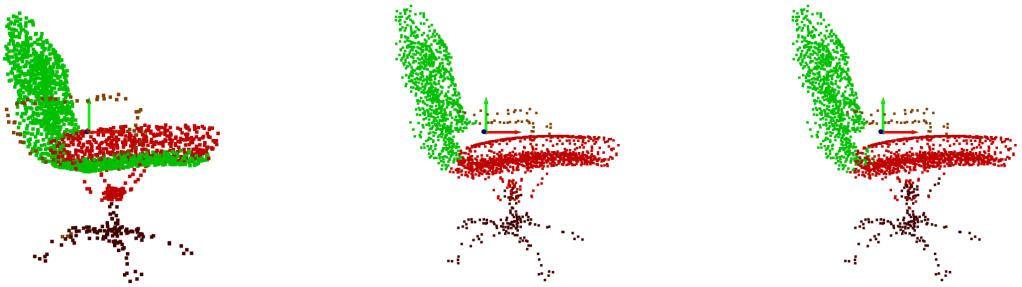
	Acc		Avg Acc		IoU		Duration (in sec)	
Model Name	Test 1	Test 2	Test 1	Test 2	Test 1	Test 2	Test 1	Test 2
PointNet	0.92	0.92	0.79	0.77	0.83	0.83	0.65	0.66
PointNet++	0.92	0.92	0.82	0.79	0.84	0.83	31.83	31.46
DGCNN (k=10)	0.92	0.93	0.74	0.73	0.81	0.81	0.79	0.8
DGCNN (k=25)	0.94	0.93	0.79	0.78	0.84	0.83	0.82	0.88
DGCNN (k=40)	0.92	0.81	0.7	0.55	0.81	0.62	0.81	0.82

Table 5.2: Comparison of the models on Test 1: untransformed test dataset and Test 2: transformed test dataset

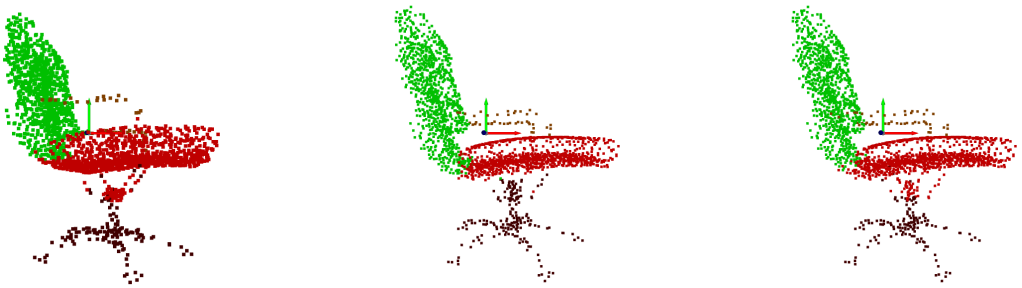
Table 5.3 provides the per object IoU obtained for during training and testing stage of KPConv. KPConv was trained for only 150 epochs instead of planned 300 as the computation time exceeded limit of 3 days.

Object	Training IoU	Testing IoU
Air	84.4	32.5
Bag	85.8	42.0
Cap	85.6	51.8
Car	80.2	30.1
Chair	90.9	38.4
Earphones	77.6	25.1
Guitar	92.4	29.7
Knife	88.7	33.6
Lamp	81.7	28.4
Laptop	95.9	57.7
Motorbike	74.6	23.1
Mug	95.6	78.1
Pistol	84.8	48.0
Rocket	66.5	56.1
Skateboard	81.0	43.7
Table	83.5	29.1
Mean	**84.3**	**40.5**

Table 5.3: Object-wise training and testing IoU's for KPConv on ShapeNetPart dataset

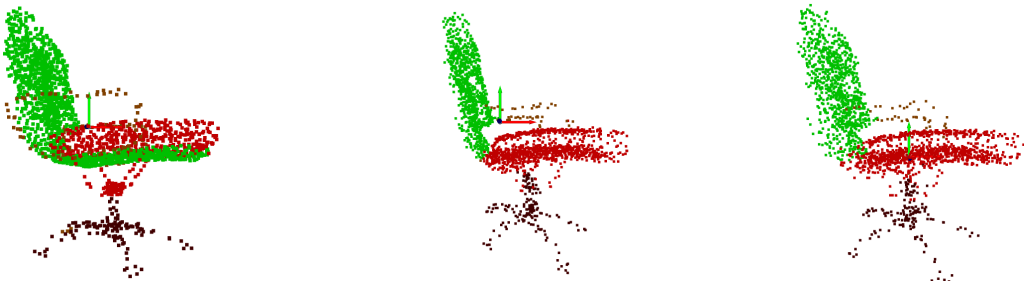


(a) True labels **(b)** Model: PointNet **(c)** Model: PointNet++

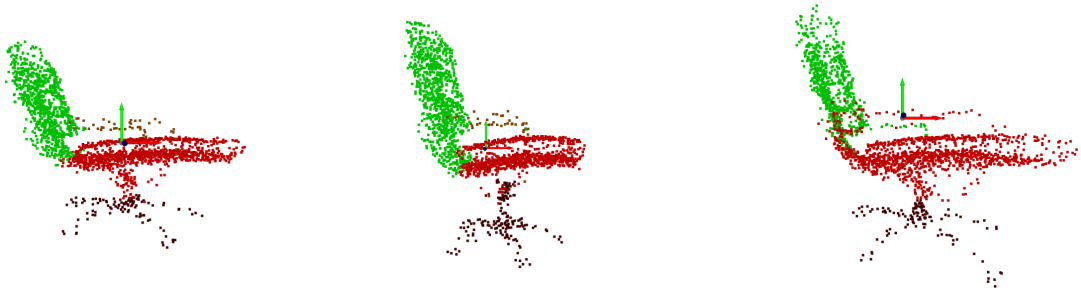


(d) Model: DGCNN (k=10) **(e)** Model: DGCNN (k=25) **(f)** Model: DGCNN (k=40)

Figure 5.2: Transformed view of point cloud with color information modified based on true labels and predicted labels for Test 1 of object 1



(a) True labels **(b)** Model: PointNet **(c)** Model: PointNet++



(d) Model: DGCNN (k=10) **(e)** Model: DGCNN (k=25) **(f)** Model: DGCNN (k=40)

Figure 5.3: Transformed view of point cloud with color information modified based on true labels and predicted labels for Test 2 of object 1

(a) True labels

(b) Model: PointNet

(c) Model: PointNet++

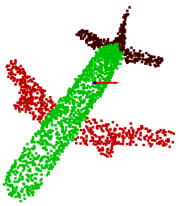
(d) Model: DGCNN (k=10)

(e) Model: DGCNN (k=25)

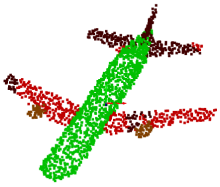
(f) Model: DGCNN (k=40)

Figure 5.4: Transformed view of point cloud with color information modified based on true labels and predicted labels for Test 1 of object 2

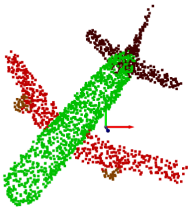
(a) True labels



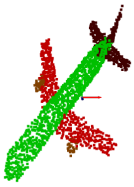
(b) Model: PointNet



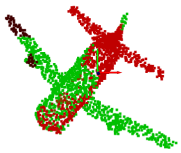
(c) Model: PointNet++



(d) Model: DGCNN (k=10)



(e) Model: DGCNN (k=25)

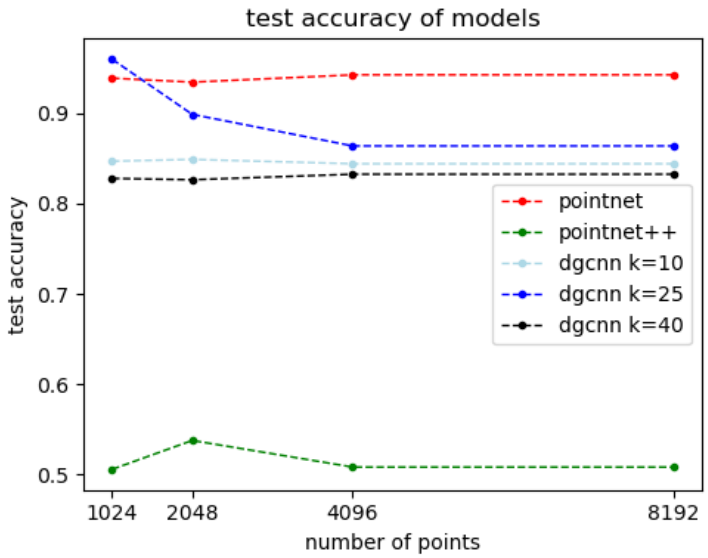


(f) Model: DGCNN (k=40)

Figure 5.5: Transformed view of point cloud with color information modified based on true labels and predicted labels for Test 2 of object 2

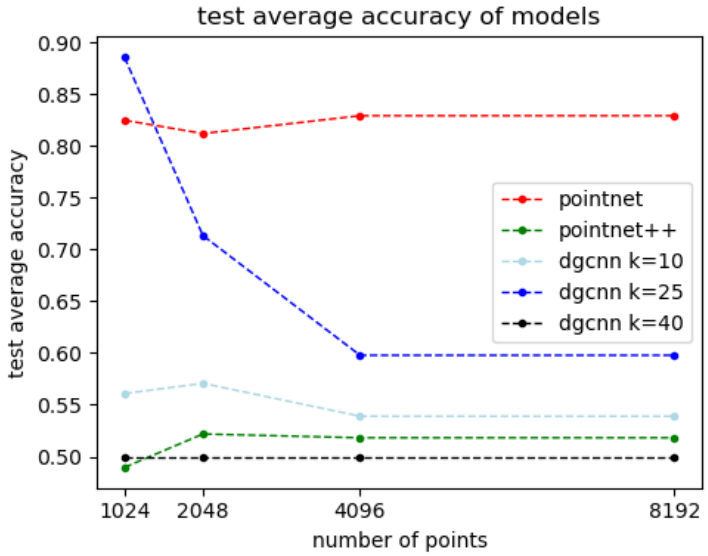
5.2 Results on Collected Test Data

In this section the models are tested on the data collected using the data collection process mentioned in the section 4.2.2. There are two objects here both belonging to the same object category of "Mug". Yet both the object have different properties. Test object 1 was mentioned in the section 4.2.2 and can be seen in figure: 4.1e, the other short more curvish "Mug" is our Test object 2 and can be seen in 5.8. The same models were tested against different number of input points (1024, 2048, 4096, 8129). Results from KPConv are not available due to technical problems. Figure: 5.6 represent different evaluation metrics used for quantizing the performance of the models. Figure: 5.6d represents the amount of time taken by a model to perform an inference/prediction of the labels. Figures {5.9, 5.7, **??**} and {5.10, 5.8, 5.12} provide visualizations of ground truth vs predictions from models for different number of input points for Test Object 1 and for Test Object 2 respectively.

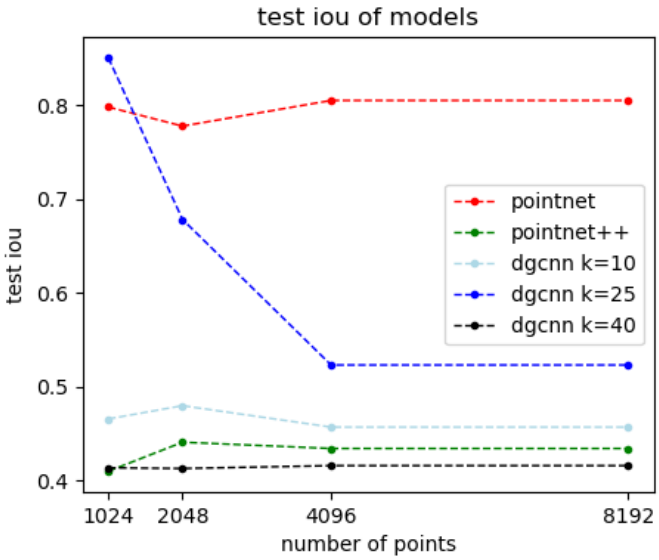


(a) Test accuracy on scanned data for varying input points

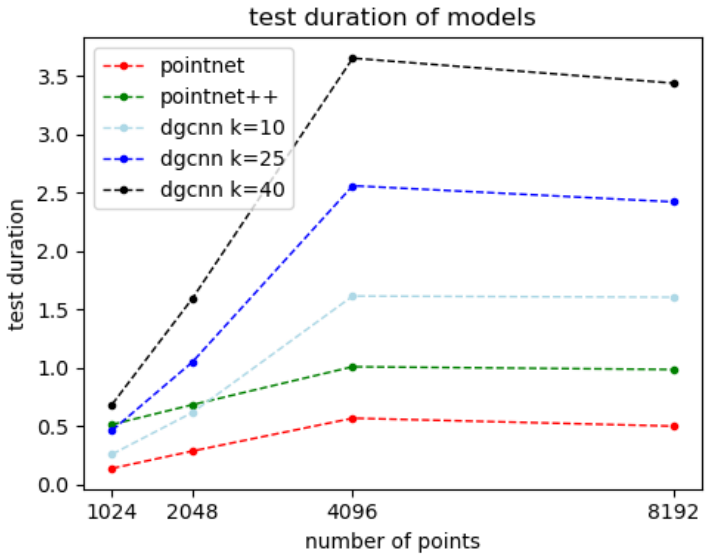
Figure 5.6: Testing models on scanned data with varying number of input points



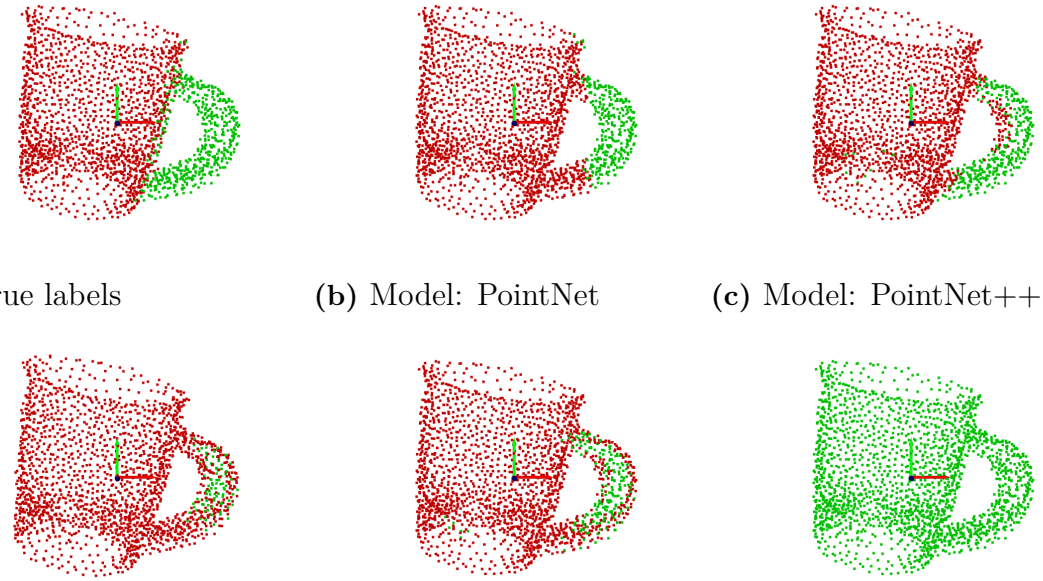
(b) Test average accuracy (average per class) on scanned data for varying input points



(c) Test IoU's on scanned data for varying input points



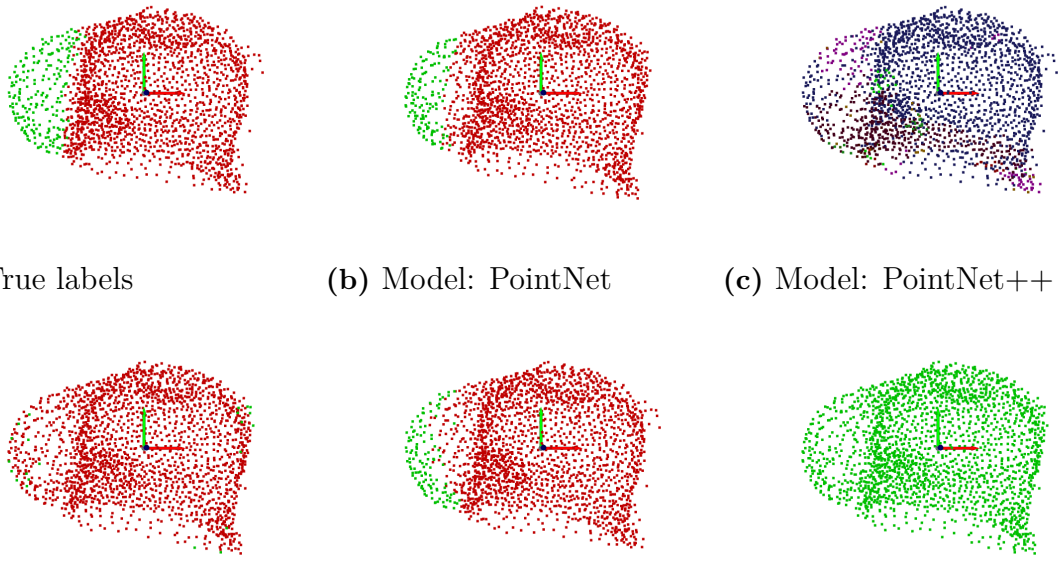
(d) Test duration on scanned data for varying input points



(a) True labels **(b)** Model: PointNet **(c)** Model: PointNet++

(d) Model: DGCNN (k=10) **(e)** Model: DGCNN (k=25) **(f)** Model: DGCNN (k=40)

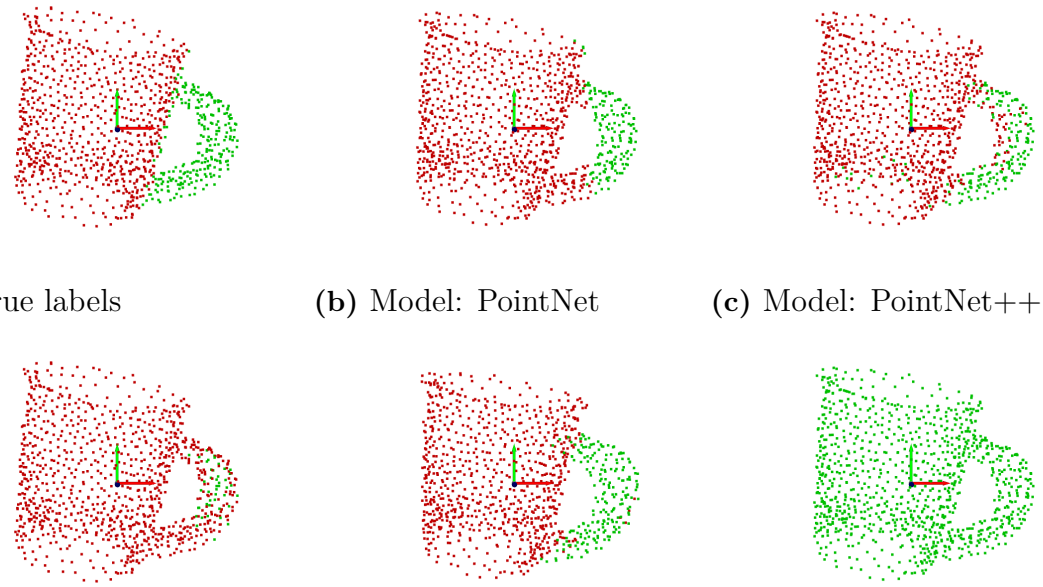
Figure 5.7: Untransformed view of point cloud with color information modified based on true labels and predicted labels for test object 1 (Number of points = 2048)



(a) True labels **(b)** Model: PointNet **(c)** Model: PointNet++

(d) Model: DGCNN (k=10) **(e)** Model: DGCNN (k=25) **(f)** Model: DGCNN (k=40)

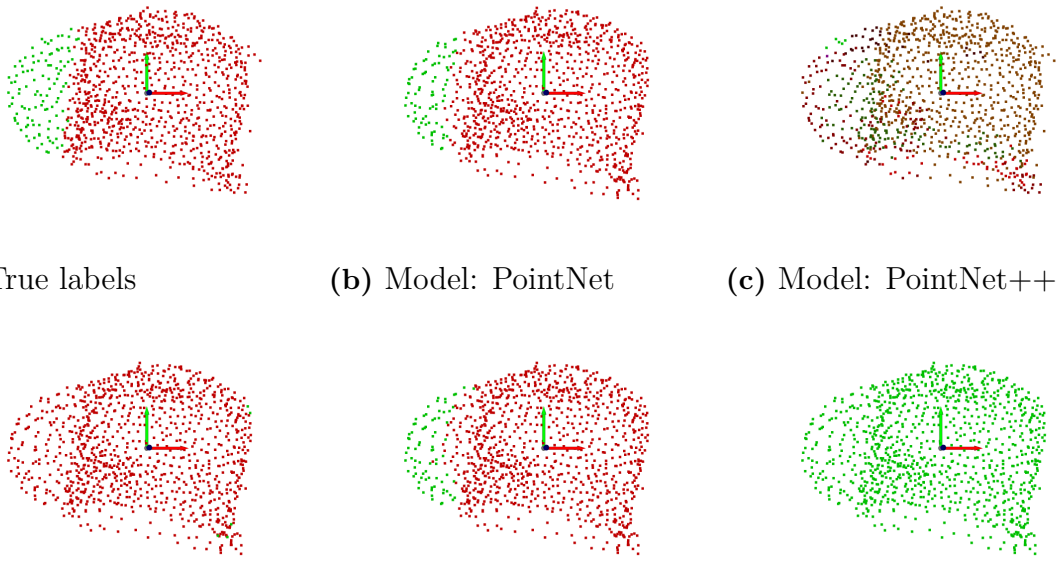
Figure 5.8: Untransformed view of point cloud with color information modified based on true labels and predicted labels for test object 2 (Number of points = 2048)



(a) True labels **(b)** Model: PointNet **(c)** Model: PointNet++

(d) Model: DGCNN (k=10) **(e)** Model: DGCNN (k=25) **(f)** Model: DGCNN (k=40)

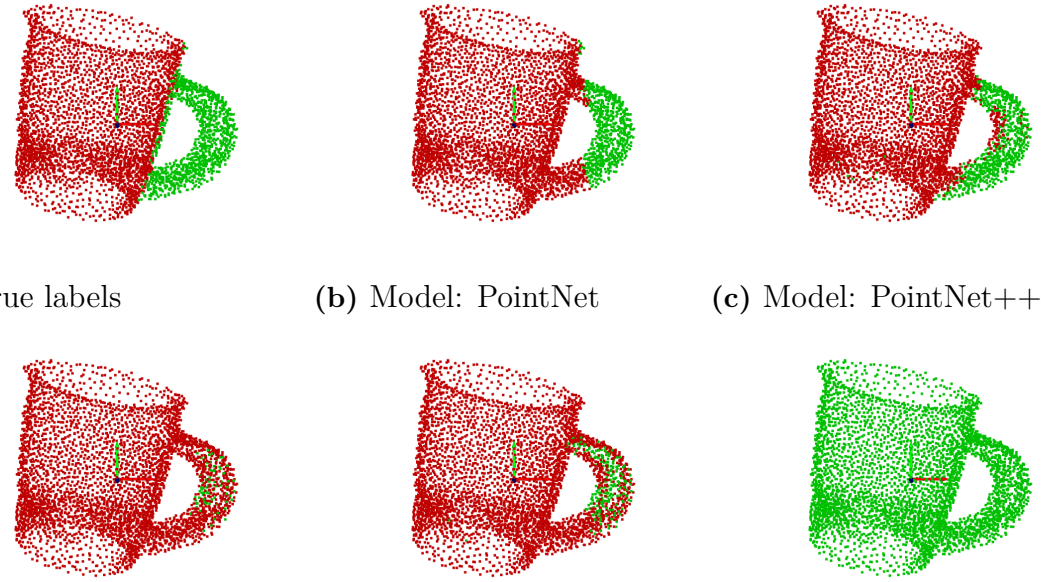
Figure 5.9: Untransformed view of point cloud with color information modified based on true labels and predicted labels for test object 1 (Number of points = 1024)



(a) True labels **(b)** Model: PointNet **(c)** Model: PointNet++

(d) Model: DGCNN (k=10) **(e)** Model: DGCNN (k=25) **(f)** Model: DGCNN (k=40)

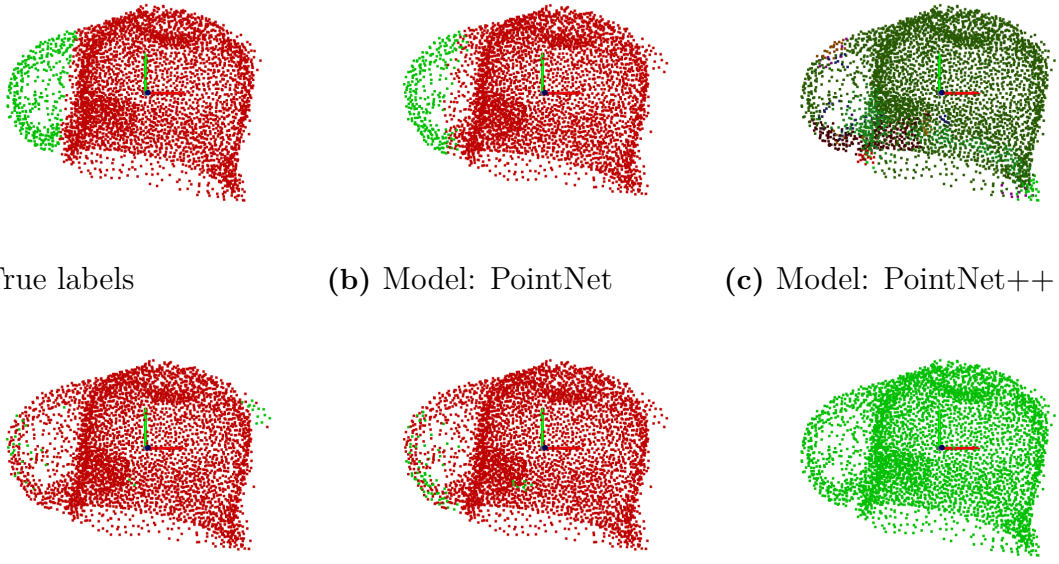
Figure 5.10: Untransformed view of point cloud with color information modified based on true labels and predicted labels for test object 2 (Number of points = 1024)



(a) True labels **(b)** Model: PointNet **(c)** Model: PointNet++

(d) Model: DGCNN (k=10) **(e)** Model: DGCNN (k=25) **(f)** Model: DGCNN (k=40)

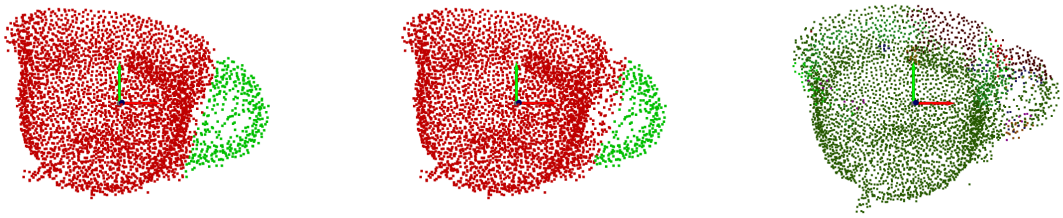
Figure 5.11: Untransformed view of point cloud with color information modified based on true labels and predicted labels for test object 1 (Number of points = 4096)



(a) True labels **(b)** Model: PointNet **(c)** Model: PointNet++

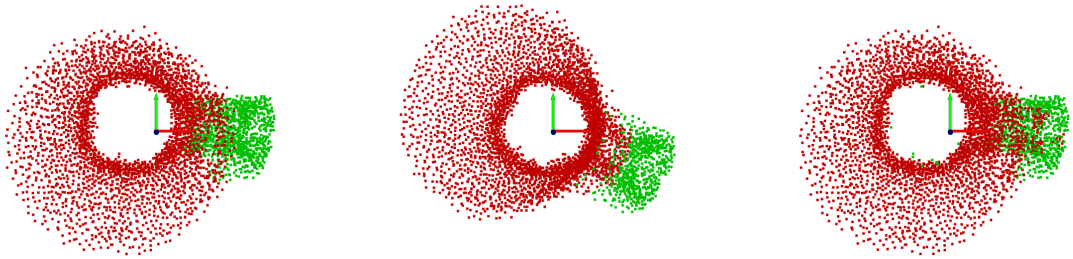
(d) Model: DGCNN (k=10) **(e)** Model: DGCNN (k=25) **(f)** Model: DGCNN (k=40)

Figure 5.12: Untransformed view of point cloud with color information modified based on true labels and predicted labels for test object 2 (Number of points = 4096)



(a) True labels **(b)** Model: PointNet **(c)** Model: PointNet++

Figure 5.13: Orthogonal View of point cloud with color information modified based on true labels and predicted labels for test object 1 (Number of points = 4096)



(a) True labels **(b)** Model: PointNet **(c)** Model: PointNet++

Figure 5.14: Top View of point cloud with color information modified based on true labels and predicted labels for test object 2 (Number of points = 4096)

6

Discussion

6.1 Invariance to geometric transformations

We recall the fact that during the training stage, the point clouds were not subjected to any kind of transformations. However, for testing, the test dataset was subjected to random translation, rotation, and jittering. However due to the data collection process, the scanned objects were inherently were translated and rotated. This is because the object was kept between 0.35 - 0.5 cm from the camera origin which was (0,0,0). and in the case of rotation, we can see that for object 1, the bottom of the mug is not parallel to the XZ-plane but is slightly oriented, whereas for object 2 the entire point cloud is not just flipped with the bottom of the mug being on top but the plane representing the bottom point cloud is not aligned with XZ-plane. Comparing the figures: 5.2 with 5.3 and 5.4 with 5.5 we can see that the overall segmentation results are quite good unless the distortion level is too high as in case of 5.5f. From table: 5.2 we can see that the there is less than 5% change between Test 1 and Test 2 for any of the metrics of all the models with DGCNN (k=40) being an exception. Models: PointNet, PointNet++ and DGCNN (k=25) have the best results on the test dataset. Looking results from the scanned data, from the figures 5.7, 5.9 and 5.11 for object 1 and 5.8, 5.10 and 5.12 for object 2, one can clearly see that PointNet outperforms all the other models irrespective of the number of input points or their orientation.

6.2 Input point cloud structure

6.2.1 Varying the number of input points

The number of input points is an important parameter, as we would ideally like the point cloud to retain as much information as possible, so that it can then be used for further applications. From figures:5.6a and 5.6d we can see that the input point cloud size directly affects the time needed by a model to make an inference. PointNet again outperforms all the other models with an inference time of around 0.5 sec for 8192 points while consistently delivering accuracy above 90% for all input point sizes.

6.2.2 Missing points and outliers

Object 1 was scanned with the bottom of the mug place on a box, recalling the test data collection process in 4.2.2, we the RANSAC-based plane segmentation removed the entire flat surface represented by the box. This eventually also removed the bottom of the mug. While no such bottomless mug was part of the training dataset, from the figure 5.14 we can see that the results have been quite satisfcatory for PointNet and PointNet++. Figures 5.11b and 5.11c offer an orthogonal view of the same result for PointNet and PointNet++ respectively. Object 2 has more prominient outliers present suchs as points in between the handle and the body of the mug, and points near the bottom of the mug. These points ideally should not exist but they have not been removed, in order to test the models against such data. Clearly PointNet is the only model to classify the points correctly while maintaining a good accuracy.

6.3 Drawbacks

6.3.1 Mislabelling of true data

All the models in this thesis are trained on the ShapeNetPart dataset, but the dataset itself constains wrong labels for certain parts of objects. One of the examples for this is presented in figure 5.5a, where we can see that there is an engine present under both the wings but it is labelled same as the wing and not as a different part. This type of mislabelling poses a problem for our models as they are supervised models. The model which suffers the most in this case is PointNet as it does not have any local feature aggregation method built in it. However, from figure 5.4 we can see that all the DGCNN model variants, PointNet++ and KPConv can identify the engine which PointNet clearly cannot.

6.3.2 Scaling issues

The results on the transformed version of the test dataset show that the models have certain resilience towards transformations like translations, rotations and jittering. However, they do not perform good if the scaling factor of the point cloud is different. Depending on the point cloud accquisition process, one may have to rescale the point cloud to match the scale of the training data. The results presented here are obtained after manually rescaling the collected the test data.

7
Conclusion

Deep learning-based methods have shown remarkable performance in point cloud segmentation. The drawback of these methods lies in the amount of time and resources spent in capturing and annotating the data, training the models, and optimizing the training process. However, they do bring in a certain value, as one does not need to handcraft features anymore but instead makes use of data and computational power to create features from the supplied data. Part segmentation is still a very challenging task compared to other segmentation tasks like instance and semantic segmentation. There exist two main bottlenecks for part segmentation, the first is the availability of datasets i.e. apart from ShapeNetPart and PartNet (part of ShapeNet project) there does not exist any other dataset. The second bottleneck is that most of these models are supervised models meaning one must provide proper labels along with the training data.

The results of PointNet, PointNet++, DGCNN and KPConv on the ShapeNetPart dataset do prove their capabilities to create features from a given point cloud but the implementation of PointNet and DGCNN was the easiest compared to PointNet++ and KPConv. Reflecting back at the training duration of each of the models and the results they have produced, it is more economical and practical to use PointNet, followed by DGCNN for solving a particular task. Especially for solving the task of part segmentation, it would be beneficial, both economically and result-wise, to make improvements or build new models based on PointNet and DGCNN.

While each model has their own merits and limitations, I believe PointNet and DGCNN are quite useful as they are ideal candidates to form a basis or draw inspiration from, for developing unsupervised models in the future like in [8]. PointNet is a surprisingly simple model with only limitation of not having any type of local feature aggregation and yet delivers amazing results. DGCNN has more customizability when it comes to change of model parameters but overfits the data very fast, making it difficult to build deeper networks. Further research in the features learned by PointNet as done in [13] and better graph regularization methods will contribute to the success of building more efficient networks. There are many directions in which these models can be optimized like better point cloud sampling strategies [10, 15], different neighborhood selection criteria, memory-efficient point processing blocks [16] and more.

Bibliography

[1] Radu Alexandru Rosu, Peer Schütt, Jan Quenzel, and Sven Behnke. *LatticeNet: Fast Point Cloud Segmentation Using Permutohedral Lattices*. Tech. rep. 2020. DOI: 10.15607/rss.2020.xvi.006.

[2] Azady Bayu, Ari Wibisono, Hanif Arief Wisesa, Naili Suri Intizhami, Wisnu Jatmiko, and Ahmad Gamal. "Semantic Segmentation of Lidar Point Cloud in Rural Area". In: *2019 IEEE International Conference on Communication, Networks and Satellite, Comnetsat 2019 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., Aug. 2019, pp. 73–78. ISBN: 9781728137957. DOI: 10.1109/COMNETSAT.2019.8844074.

[3] Dena Bazazian and Dhananjay Nahata. "DCG-Net: Dynamic Capsule Graph Convolutional Network for Point Clouds". In: *IEEE Access* 8 (Oct. 2020), pp. 188056–188067. ISSN: 2169-3536. DOI: 10.1109/access.2020.3031812.

[4] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. "ShapeNet: An Information-Rich 3D Model Repository". In: (Dec. 2015). URL: http://arxiv.org/abs/1512.03012.

[5] Sungjoon Choi, Qian Yi Zhou, and Vladlen Koltun. "Robust reconstruction of indoor scenes". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 07-12-June-2015. IEEE Computer Society, Oct. 2015, pp. 5556–5565. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7299195.

[6] Nvidia Corporation. *NVIDIA V100 TENSOR CORE GPU*. Tech. rep. 2020. URL: www.nvidia.com/v100.

[7] Ahmad Gamal, Ari Wibisono, Satrio Bagus Wicaksono, Muhammad Alvin Abyan, Nur Hamid, Hanif Arif Wisesa, Wisnu Jatmiko, and Ronny Ardhianto. "Automatic LIDAR building segmentation based on DGCNN and euclidean clustering". In: *Journal of Big Data* 7.1 (Dec. 2020). ISSN: 21961115. DOI: 10.1186/s40537-020-00374-x.

[8] Xiang Gao, Wei Hu, and Guo Jun Qi. "Graphter: Unsupervised learning of graph transformation equivariant representations via auto-encoding node-wise transformations". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2020. DOI: 10.1109/CVPR42600.2020.00719.

[9] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. "Deep learning for 3D point clouds: A survey". In: *arXiv* (Dec. 2019), pp. 1–27. ISSN: 23318422. DOI: 10.1109/tpami.2020.3005434. URL: https://github.com/QingyongHu/SoTA-Point-Cloud..

[10] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. "Randla-Net: Efficient semantic segmentation of large-scale point clouds". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Nov. 2020), pp. 11105–11114. ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.01112. URL: http://arxiv.org/abs/1911.11236.

[11] *ICP registration — Open3D latest (db7f79c) documentation*. 2021. URL: http://www.open3d.org/docs/latest/tutorial/pipelines/icp_registration.html#Point-to-plane-ICP.

[12] Hyeonho Jeong, Hyosung Hong, Gyuha Park, Mooncheol Won, Mingyu Kim, and Hoyeong Yu. "Point cloud segmentation of crane parts using dynamic graph CNN for crane collision avoidance". In: *Journal of Computing Science and Engineering* 13.3 (2019). ISSN: 20938020. DOI: 10.5626/JCSE.2019.13.3.99.

[13] Mor Joseph-Rivlin, Alon Zvirin, and Ron Kimmel. "Momen(e)t: Flavor the Moments in Learning to Classify Shapes". In: *Proceedings - 2019 International Conference on Computer Vision Workshop, ICCVW 2019* (Dec. 2018), pp. 4085–4094. URL: http://arxiv.org/abs/1812.07431.

[14] Martin Kiefel, Varun Jampani, and Peter V. Gehler. "Permutohedral Lattice CNNs". In: *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings* (Dec. 2014). URL: http://arxiv.org/abs/1412.6618.

[15] Itai Lang, Asaf Manor, and Shai Avidan. "SampleNet: Differentiable Point Cloud Sampling". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Dec. 2019), pp. 7575–7585. URL: http://arxiv.org/abs/1912.03663.

[16] Eric Tuan Le, Iasonas Kokkinos, and Niloy J. Mitra. "Going deeper with lean point networks". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (July 2020), pp. 9500–9509. ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.00952. URL: http://arxiv.org/abs/1907.00960.

[17] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. "PointCNN: Convolution On $\mathcal{X}$-Transformed Points". In: *arXiv* (Jan. 2018), pp. 1–11. URL: http://arxiv.org/abs/1801.07791.

[18] Haibin Ling and David W. Jacobs. "Shape classification using the inner-distance". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.2 (Feb. 2007), pp. 286–299. ISSN: 01628828. DOI: 10.1109/TPAMI.2007.41.

[19] *Multiway registration — Open3D latest (db7f79c) documentation*. 2021. URL: http://www.open3d.org/docs/latest/tutorial/pipelines/multiway_registration.html.

[20] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. *DeepSDF: Learning continuous signed distance functions for shape representation*. Tech. rep. 2019.

[21] Regina Pohle-Fröhlich, Aaron Bohm, Peer Ueberholz, Maximilian Korb, and Steffen Goebbels. "Roof segmentation based on deep neural networks". In: *VISIGRAPP 2019 - Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. Vol. 4. SciTePress, 2019, pp. 326–333. ISBN: 9789897583544. DOI: 10.5220/0007343803260333.

[22] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. "PointNet: Deep learning on point sets for 3D classification and segmentation". In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. Vol. 2017-January. Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 77–85. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.16. URL: https://arxiv.org/abs/1612.00593v2.

[23] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Advances in Neural Information Processing Systems* 2017-December (June 2017), pp. 5100–5109. URL: http://arxiv.org/abs/1706.02413.

[24] Szymon Rusinkiewicz and Marc Levoy. "Efficient variants of the ICP algorithm". In: *Proceedings of International Conference on 3-D Digital Imaging and Modeling, 3DIM* (2001), pp. 145–152. ISSN: 15506185. DOI: 10.1109/IM.2001.924423.

[25] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. "Fast Point Feature Histograms (FPFH) for 3D registration". In: Institute of Electrical and Electronics Engineers (IEEE), Aug. 2009, pp. 3212–3217. DOI: 10.1109/robot.2009.5152473.

[26] Christoph Sager, Patrick Zschech, and Niklas Kühl. "labelCloud: A Lightweight Domain-Independent Labeling Tool for 3D Object Detection in Point Clouds". In: (Mar. 2021). URL: http://arxiv.org/abs/2103.04970.

[27] M. Soilán, A. Nóvoa, A. Sánchez-Rodríguez, B. Riveiro, and P. Arias. "Semantic Segmentation of Point Clouds with Pointnet and Kpconv Architectures Applied to Railway Tunnels". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Vol. 5. 2. Copernicus GmbH, Aug. 2020, pp. 281–288. DOI: 10.5194/isprs-annals-V-2-2020-281-2020.

[28] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming Hsuan Yang, and Jan Kautz. "SPLATNet: Sparse Lattice Networks for Point Cloud Processing". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2530–2539. ISBN: 9781538664209. DOI: 10.1109/CVPR.2018.00268.

[29] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G Learned-Miller. "Multi-view Convolutional Neural Networks for 3D Shape Recognition". In: *CoRR* abs/1505.00880 (2015). URL: http://arxiv.org/abs/1505.00880.

[30] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. "Tangent Convolutions for Dense Prediction in 3D". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (July 2018), pp. 3887–3896. URL: http://arxiv.org/abs/1807.02443.

[31] Hugues Thomas, Charles R. Qi, Jean Emmanuel Deschaud, Beatriz Marcotegui, Francois Goulette, and Leonidas Guibas. "KPConv: Flexible and deformable convolution for point clouds". In: *Proceedings of the IEEE International Conference on Computer Vision*. Vol. 2019-Octob. IEEE, Oct. 2019, pp. 6410–6419. ISBN: 9781728148038. DOI: 10.1109/ICCV.2019.00651. URL: https://ieeexplore.ieee.org/document/9010002/.

[32] *Vera - C3SE*. URL: https://www.c3se.chalmers.se/about/Vera/.

[33] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. "SGPN: Similarity group proposal network for 3D point cloud instance segmentation". In: *arXiv* (2017). ISSN: 23318422.

[34] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. "Dynamic graph Cnn for learning on point clouds". In: *ACM Transactions on Graphics* 38.5 (2019). ISSN: 15577368. DOI: 10.1145/3326362.

[35] Chen Yang and Gérard Medioni. "Object modelling by registration of multiple range images". In: *Image and Vision Computing* 10.3 (Apr. 1992), pp. 145–155. ISSN: 02628856. DOI: 10.1016/0262-8856(92)90066-C.

[36] Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. "SyncSpecCNN: Synchronized Spectral CNN for 3D Shape Segmentation". In: *CoRR* abs/1612.00606 (2016). URL: http://arxiv.org/abs/1612.00606.

[37] Fenggen Yu, Kun Liu, Yan Zhang, Chenyang Zhu, and Kai Xu. "PartNet: A Recursive Part Decomposition Network for Fine-grained and Hierarchical Shape Segmentation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2019-June (Mar. 2019), pp. 9483–9492. URL: http://arxiv.org/abs/1903.00709.

[38] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". In: (Jan. 2018). URL: http://arxiv.org/abs/1801.09847.

[39] Yin Zhou and Oncel Tuzel. "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2018. DOI: 10.1109/CVPR.2018.00472.

A

Appendix 1: Algorithms

A.1 Farthest Point Sampling

Algorithm 1: Fastest Point Sampling

Input: P: matrix representing point cloud data of size
(N points x M dimensions)
num_points: number of samples

Output: centroids: vector containing index of sampled point cloud of size
($num\_points, 1$)

/* to store index values of centroids from the input point cloud */
1 $centroids \leftarrow zeros(num\_points, 1)$
/* to store euclidean distance between a point and a sets of points */
2 $distance \leftarrow ones(N, 1) * 1e^{10}$
/* returns random integer such that $0 \leq X \leq N$ */
3 $farthest \leftarrow randint(0, N)$

for $i \leftarrow 0$ **to** num_points **by** 1 **do**
  /* Set the centroid of i^{th} point to farthest point */
4 $centroids[i, 1] \leftarrow farthest$
  /* Take the coordinate of P at the farthest point */
5 $cloud\_centroid \leftarrow P[farthest, M]$
  /* Calculate the Euclidean distance from all points in the point set to this farthest point */
6 $dist \leftarrow ((P - cloud\_centroid)^2)$
  /* Update distances to record the minimum distance of each point in the sample from all existing sample points */
7 $dist\_mask \leftarrow dist < distance$
8 $distance[dist\_mask] \leftarrow dist[dist\_mask]$
  /* Find the farthest point from the updated distances matrix, and use it as the farthest point for the next iteration */
9 $farthest \leftarrow max(distance)$
10 **return** $centroids$

A.2 Ball Query Search

Ball query search [23] is a type of nearest neighborhood search very close to the k-NN approach. The idea behind the ball query search is that given a query point p_s which lies in the point cloud P, we shall find all points $P^{'}$ such that the euclidean distance between p_s and every point of P is less than a set radius r. To compute p_s, farthest point sampling is used to assure reception over the entire point cloud P. The advantage of using ball query search over k-NN is that the ball query shall find all points in a given radius r while k-NN shall find only k fixed points. However for computational reasons, in the implementation, there is an upper limit for the total number of points sampled using ball query. Following is the pseudocode for ball query search:

Algorithm 2: Ball Query Search

Input: r: radius of local region
$sample\_limit$: maximum number of samples in a local region
P: input point matrix of size (N points x M dimensions)
Q: query point matrix of size (S query points x M dimensions)

Output: $group\_idx$: index of grouped points of size ($S \times sample\_limit$)

$group\_idx = torch.arange(N).view(1, 1, N).repeat([B, S, 1])$

/* sqrdists: [K, N] Store the Euclidean distance between the center point Q and all points P */

$sqrdists = square\_distance(Q, P)$

/* Find all distances greater than r^2, its $group\_idx$ is directly set them to N */

$group\_idx[sqrdists > r^2] = N$

/* Sort the array in an ascending order and select the first $sample\_limit$ number of points */

$group\_idx[:,] = sort(group\_idx)[:, : sample\_limit]$

/* Considering that there may be points in the previous num_sample points that are assigned N (ie, less than num_sample points in the spherical area), this point needs to be discarded, and the first point can be used instead. */

/* $group\_first$: [B, S, k], actually copy the value of the first point in $group\_idx$ to the dimension of [B, S, K], which is convenient for subsequent replacement. */

$group\_first = group\_idx[:, :, 0].view(B, S, 1).repeat([1, 1, num\_sample])$

/* Find the point where $group\_idx$ is equal to N */

$mask = group\_idx == N$

/* Replace the value of these points with the value of the first point */

$group\_idx[mask] = group\_first[mask]$

return group_idx

B

Appendix 2: Test Data Collection Pipeline

B.1 RANSAC-based Plane segmentation

To remove the top of the box on which the object is place, we make use of RANSAC to find a plane which fits the largest number of points and eventually represents the top of the box. To achieve this we make use of Open3D's segment_plane function as mentioned in [11]. It has the following three parameters:

- A distance threshold which defines the maximum distance a point can have to an estimated plane, for it to be considered an inlier i.e. a point within the estiamted plane.
- The number of points that are randomly sampled to estimate a plane.
- The number of iterations which states how often a random plane is to be sampled and verified.

The function then returns the coefficients of the plane as (a, b, c, d), such that for each point (x, y, z) on the plane we have $ax + by + cz + d = 0$ and a list of indices of the points which lie within the plane. Making use of this list of inlier indices, we can remove the points which belong to the top of the box. The plane removal takes place for all the point clouds, but the parameters for the function are set only once. To find the optimum parameters effectively, a visualization of the segmented point cloud is shown using default parameter values and if the results are not satisfactory, new parameters can be entered by the user, followed by a new visualization. This process takes place in an iterative fasion till the user decides to stop the program, hence setting the argument values of the function to be used for all the remaining point clouds. If needed the plane segmentation function can also be called only once or be used in the above mentioned way to process a set of point clouds.

B.2 Point cloud fusion

B.2.1 ICP registration

The main algorithm used to fuse multiple point clouds here is called Multiway registration which essentially uses a point cloud registration algorithm like Iterative Closest Point (ICP) as the base algorithm. ICP takes in two point clouds (source and target) as an input, and an initial transformation that might roughly align the

source point cloud to the target point cloud. The output of ICP is a transformation that aligns the two point clouds. There many variations of ICP available as mentioned in [24], the one used here is called as point-to-plane ICP [35] and is implemented via Open3D [11].

The point-to-plane ICP algorithm iterates over two steps:

1. To find the correspondence set $\mathbf{C_s} = \{(p_i, p_j)\}$ from target point cloud $\mathbf{P_j}$, and source point cloud $\mathbf{P_i}$ which is transformed with the transformation matrix $\mathbf{T_{i,j}}$. Here p_i and p_j are points from the source and the target point clouds respectively.
2. Update the transformation matrix $\mathbf{T_{i,j}}$ by minimizing an objective function $\mathbf{E}(\mathbf{T})$ defined over the correspondence set $\mathbf{C_s}$

.

The objective function $\mathbf{E}(\mathbf{T})$ is defined as follows

$$E(\mathbf{T}) = \sum_{(p,q)\in K} ((p - Tq) \cdot n_p)^2 \tag{B.1}$$

where, n_p is the normal of point p.

B.2.2 Multiway registration

Multiway registration is a process used to find transformations, $\{\mathbf{T_i}\}$ which for a set of point clouds, $\{\mathbf{P_i}\}$ such that the transformed point clouds $\{\mathbf{T_i P_i}\}$ are aligned in a common space. Multiway registration is achieved via the pose graph optimization and implemented using the functions in Open3D [19]. Pose graph optimization is a two step process in which one first constructs a pose graph and then optimizes.

A pose graph is just a list consisting of nodes and edges. Every node consinsts of a point cloud $\mathbf{P}$ and a pose (transformation) matrix $\mathbf{T}$, which transforms $\mathbf{P}$ into the common global space. By default, the space of the first point cloud $\mathbf{P_0}$ from the input point cloud set is used as the common global space and $\mathbf{T_0}$ is set as an identity matrix. The concept of multiway registration assumes that neighboring nodes will have a large overlap and hence, can be registered using the point-to-plane ICP algorithm. The objective of the edge is to connect two different nodes that have overlap between them. Each edge consists of a pose matrix $\mathbf{T_{i,j}}$ which can align a source point cloud $\mathbf{P_i}$ to a target point cloud $\mathbf{P_j}$.

The alogirthm then begins to construct a pose graph by comparing all the points clouds with every other point cloud, the registration process between two point clouds is called as pairwise registration. From [5] one finds that pairwise registration is error-prone and false positives outnumber true positives. To overcome this problem, they suggest partitioning the pose graph edges into two types. The first type of edge is called odometry edge, which denotes a connection between temporally close, neighboring nodes. The second type of edge is called loop closure edge, which denotes connectiong between any non-neighboring nodes. Further, a local and global registration algorithm is used for finding the transformation matrix for

odometry edges and loop closure edges respectively. After running the entire process, the end result is a fused point cloud in a common global space. This fused point cloud then be further processed as required and finally annotated using [26].

C

Appendix 3: Intel Realsense L515 Operating Modes

Operating models/streaming profiles supported by the Depth Sensor of L515:

No.	Stream	Resolution	FPS	Format
1a	Confidence	1024x768	30Hz	RAW8
1b	Confidence	640x480	30Hz	RAW8
1c	Confidence	320x240	30Hz	RAW8
2a	Infrared	1024x768	30Hz	Y8
2b	Infrared	640x480	30Hz	Y8
2c	Infrared	320x240	30Hz	Y8
3a	Depth	1024x768	30Hz	Z16
3b	Depth	640x480	30Hz	Z16
3c	Depth	320x240	30Hz	Z16

Table C.1: Stream Profiles supported by the L515 Depth Sensor

Operating models/streaming profiles supported by the Motion Module of L515:

No.	Stream	Resolution	FPS	Format
1a	Accel	N/A	400Hz	MOTION_XYZ32F
1b	Accel	N/A	200Hz	MOTION_XYZ32F
1c	Accel	N/A	100Hz	MOTION_XYZ32F
2a	Gyro	N/A	400Hz	MOTION_XYZ32F
2b	Gyro	N/A	200Hz	MOTION_XYZ32F
2c	Gyro	N/A	100Hz	MOTION_XYZ32F

Table C.2: Stream Profiles supported by the Motion Module

Operating models/streaming profiles supported by the RGB Camera of L515:

No.	Stream	Resolution	FPS	Format	No.	Stream	Resolution	FPS	Format
1	Color	1920x1080	30Hz	RGB8	34	Color	1280x720	15Hz	RGBA8
2	Color	1920x1080	30Hz	Y16	35	Color	1280x720	15Hz	BGR8
3	Color	1920x1080	30Hz	BGRA8	36	Color	1280x720	15Hz	YUYV
4	Color	1920x1080	30Hz	RGBA8	37	Color	1280x720	6Hz	RGB8
5	Color	1920x1080	30Hz	BGR8	38	Color	1280x720	6Hz	Y16
6	Color	1920x1080	30Hz	YUYV	39	Color	1280x720	6Hz	BGRA8
7	Color	1920x1080	15Hz	RGB8	40	Color	1280x720	6Hz	RGBA8
8	Color	1920x1080	15Hz	Y16	41	Color	1280x720	6Hz	BGR8
9	Color	1920x1080	15Hz	BGRA8	42	Color	1280x720	6Hz	YUYV
10	Color	1920x1080	15Hz	RGBA8	43	Color	960x540	60Hz	Y16
11	Color	1920x1080	15Hz	BGR8	44	Color	960x540	60Hz	BGRA8
12	Color	1920x1080	15Hz	YUYV	45	Color	960x540	60Hz	RGBA8
13	Color	1920x1080	6Hz	RGB8	46	Color	960x540	60Hz	BGR8
14	Color	1920x1080	6Hz	Y16	47	Color	960x540	60Hz	YUYV
15	Color	1920x1080	6Hz	BGRA8	48	Color	960x540	30Hz	RGB8
16	Color	1920x1080	6Hz	RGBA8	49	Color	960x540	30Hz	Y16
17	Color	1920x1080	6Hz	BGR8	50	Color	960x540	30Hz	BGRA8
18	Color	1920x1080	6Hz	YUYV	51	Color	960x540	30Hz	RGBA8
19	Color	1280x720	60Hz	RGB8	52	Color	960x540	30Hz	BGR8
20	Color	1280x720	60Hz	Y16	53	Color	960x540	30Hz	YUYV
21	Color	1280x720	60Hz	BGRA8	54	Color	960x540	15Hz	RGB8
22	Color	1280x720	60Hz	RGBA8	55	Color	960x540	15Hz	Y16
23	Color	1280x720	60Hz	BGR8	56	Color	960x540	15Hz	BGRA8
24	Color	1280x720	60Hz	YUYV	57	Color	960x540	15Hz	RGBA8
25	Color	1280x720	30Hz	RGB8	58	Color	960x540	15Hz	BGR8
26	Color	1280x720	30Hz	Y16	59	Color	960x540	15Hz	YUYV
27	Color	1280x720	30Hz	BGRA8	60	Color	960x540	6Hz	RGB8
28	Color	1280x720	30Hz	RGBA8	61	Color	960x540	6Hz	Y16
29	Color	1280x720	30Hz	BGR8	62	Color	960x540	6Hz	BGRA8
30	Color	1280x720	30Hz	YUYV	63	Color	960x540	6Hz	RGBA8
31	Color	1280x720	15Hz	RGB8	64	Color	960x540	6Hz	BGR8
32	Color	1280x720	15Hz	Y16	65	Color	960x540	6Hz	YUYV
33	Color	1280x720	15Hz	BGRA8					

Table C.3: Stream Profiles supported by the RGB Camera