8-10-2020

Point Completion Networks and Segmentation of 3D Mesh

Naga Durga Harish Kanamarlapudi
nk4464@rit.edu

Point Completion Networks and Segmentation of 3D Mesh

By

Naga Durga Harish Kanamarlapudi

August 10, 2020

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

Rochester Institute of Technology

Committee Approval:

Date

Dr. Raymond Ptucha, *Advisor*
Primary Advisor- R.I.T. Dept. of Computer Engineering

Date

Dr. Alexander Loui, *Committee Member*
Committee Member – R.I.T. Dept. of Computer Engineering

Date

Dr. Guoyu Lu, *Committee Member*
Committee Member – R.I.T. Center for Imaging Science

RIT | Kate Gleason College of Engineering

Department of Computer Engineering

Acknowledgments

I would like to take this opportunity to thank my advisor Dr. Raymond Ptucha for his continual support and guidance during my master’s degree. I am thankful for Dr. Alexander Loui and Dr. Guoyu Lu for being in my thesis committee. I would like to thank my family and my close friends, without their support this journey would not have been as joyful as it was.

Abstract

Deep learning has made many advancements in fields such as computer vision, natural language processing and speech processing. In autonomous driving, deep learning has made great improvements pertaining to the tasks of lane detection, steering estimation, throttle control, depth estimation, 2D and 3D object detection, object segmentation and object tracking. Understanding the 3D world is necessary for safe end-to-end self-driving. 3D point clouds provide rich 3D information, but processing point clouds is difficult since point clouds are irregular and unordered. Neural point processing methods like GraphCNN and PointNet operate on individual points for accurate classification and segmentation results. Occlusion of these 3D point clouds remains a major problem for autonomous driving. To process occluded point clouds, this research explores deep learning models to fill in missing points from partial point clouds. Specifically, we introduce improvements to methods called deep multistage point completion networks. We propose novel encoder and decoder architectures for efficiently processing partial point clouds as input and outputting complete point clouds. Results will be demonstrated on ShapeNet dataset.

Deep learning has made significant advancements in the field of robotics. For a robot gripper such as a suction cup to hold an object firmly, the robot needs to determine which portions of an object, or specifically which surfaces of the object should be used to mount the suction cup. Since 3D objects can be represented in many forms for computational purposes, a proper representation of 3D objects is necessary to tackle this problem. Formulating this problem using deep learning problem provides dataset challenges. In this work we will show representing 3D objects in the form of 3D mesh is effective for the problem of a robot gripper. We will perform research on the proper way for dataset creation and performance evaluation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Neural networks helped to obtain state of the art results in many of the challenging tasks like image recognition, image classification, 3D point cloud classification, text understanding and speech recognition. Convolutional Neural Networks (CNNs) have replaced the traditional computer vision techniques like Histogram of Oriented Gradients (HOG) and Scale Invariant Feature Transform (SIFT) to perform many of the vision related tasks like classification, and recognition. Architectures like ResNet [33], VGGNet [56], DenseNet [57] have shown the ability of CNNs to extract the features to obtain state of the art results in many vision related tasks like classification and segmentation.

Understanding the 3D world is one of the most challenging tasks in Artificial Intelligence (AI). Robust scene understanding is required in the case of applications like end-to-end autonomous driving and robotics. 3D data is typically represented in the format of point clouds and meshes. Due to the irregularity and unordered nature of point clouds, applying CNNs is not straight forward. Many of the previous works [1, 25, 13, 4, 17, 6] used hand crafted features of point clouds, other works [36] converted point clouds to voxels and used 3D CNNs to perform tasks, some works [10, 14] used Multiview CNNs by converting point clouds to images, and spectral CNNs [11, 12] use convolutions in spectral domain. Neural point processing using PointNet [15] extracts point cloud features by operating on individual point. PointNet++ [14] improves PointNet by extracting point cloud features hierarchically using multi-scale and multi-resolution grouping. Due to its simplistic architecture and effective performance, PointNet inspired many point cloud processing techniques [16, 5, 18].

Occlusion is a major problem in the real-world LiDAR scans. For safe end-to-end self-driving, incomplete point clouds should be made complete. PCN [2] takes the partial point cloud as input and outputs the complete point cloud. In this work, we propose novel encoder and decoder architectures for deep multistage point completion networks.

For a robot gripper like suction cup, in order to hold a 3D object firmly there should be a minimum flat surface area on the 3D object for the suction cup to get a good grip. The good area can be anywhere on the 3D object. Identifying different good parts on the outer surface of the 3D object is difficult. 3D objects can be represented in different forms like point clouds, voxels and mesh. Representing 3D object with point clouds gives information of outer surface, but only in a sparse fashion. Representing 3D objects with voxels is generally limited by memory and doesn't have high resolution representation of the object surface. Further, the voxels in the center of the object are generally wasted memory. Representing 3D objects in the form of surface mesh provides more information of outer surface of 3D object and we can represent mesh in dense form due to its connectivity among different faces. So, formulating this problem as mesh segmentation helps to identify different parts of 3D object which are good a good surface for a robot gripper. Deep learning on 3D mesh data is a newer problem, [48] is the first deep learning model applied to 3D mesh data. After formulating this problem as a machine learning problem, getting the dataset suitable for this task is very difficult because there are no publicly available datasets dealing this problem. In this work, we show some methods for creating and annotating the dataset effectively and show initial results using existing deep learning architectures.

1.2 Motivation

Understanding the 3D world is very important for end-to-end autonomous driving, and occlusions create many problems as deep learning networks typically only see only a partial shape of the object. This research completes the partial shape of the object to minimize this problem. Also, for efficient point cloud registration, registration on full shapes provides better registration results instead of partial shapes. This work deals with completing a partial point cloud.

To identify good and bad faces of 3D objects to hold firmly by a robot gripper such as a suction cup, there are no existing algorithms. Converting the 3D object into 3D mesh and segmenting the mesh into good and bad faces provides information regarding various good faces for a robot gripper like suction cup to hold firmly.

1.3 Contributions

The main contributions of this thesis work can be summarized as follows:

- Experimented with different novel deep learning architectures.
- Combining the voxel wise and point wise features for better global feature vector.
- Complete shape of the input partial point cloud is optimized in multiple stages like coarse, middle and fine.
- Explored the capsule based dynamic routing architecture for the problem of point completion network.

- Experimented with self-supervised approach for the task of point completion network.
- Created MATLAB tool for dataset generation for the robot gripper task.
- Experimented with different deep learning architectures for the robot gripper task.

Chapter 2

Background

2.1 Deep Learning

Deep learning has achieved state of the art results in many computer vision related tasks like 2D and 3D object detection, mesh classification, point cloud classification and segmentation, and image captioning. Deep learning has replaced traditional computer vision techniques for feature extraction in the case of 3D data like point clouds and meshes. [36, 55] converts point clouds to voxels, [15, 16, 14] computes point-wise features and applies symmetric function like maxpooling, and [48] extracts the mesh features by considering each face as a unit.

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have become the standard method for extracting the features of images and point wise features in point clouds. CNNs typically consist of the following layers:

- Convolution layer
- Pooling layer
- Activation layer
- Fully connected layer

A convolution layer consists of multiple filters of a window size multiplied with the input pixels and performs a linear combination of all the multiplied values in that filter window. Filters in the initial layers learn low level features and filters in the later layers learn higher level features of the input data. Usually, the convolution operation will be performed in 1D, 2D or 3D. A 1D convolution is shown in Figure 1 and a 2D convolution operation is shown in Figure 2.

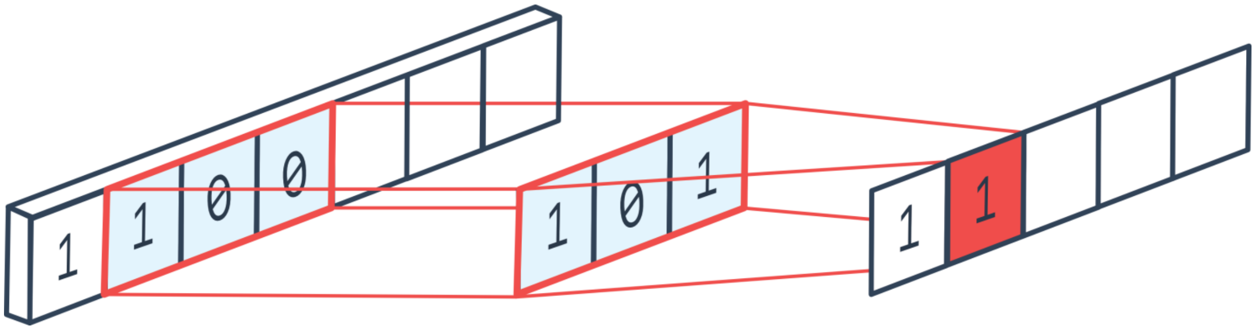


Figure 1 An example of 1D Convolution [40].

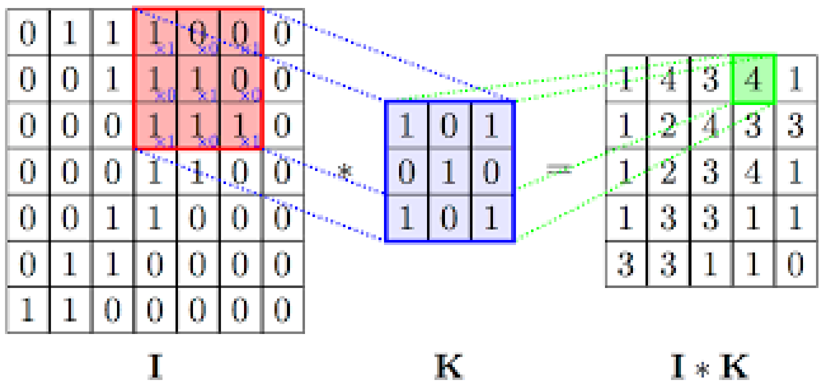


Figure 2 An example of 2D Convolution [41].

A pooling layer down samples the image depending on the type of pooling used. There are two kinds of pooling. One is max pooling as shown in Figure 3, which outputs the maximum value of the pixels in a certain pooling window. The other is average pooling as shown in Figure 4, which outputs the average value of all pixel values in a certain pooling window.

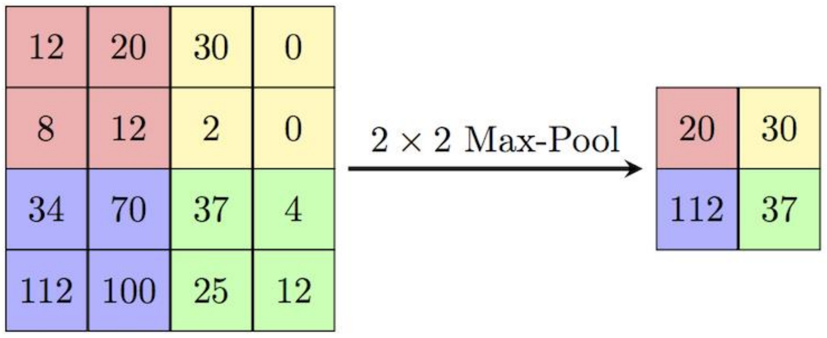


Figure 3 An example of 2D Maxpooling [42].

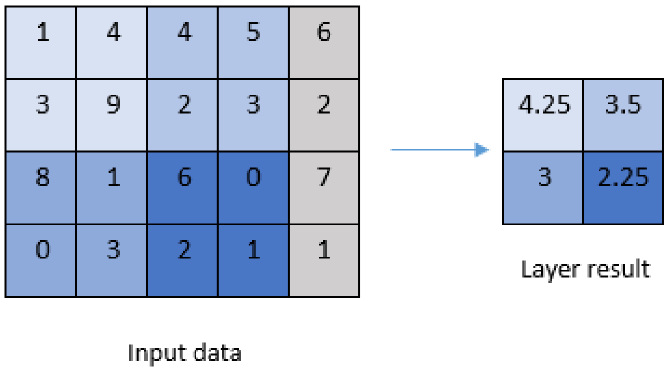


Figure 4 An example of 2D Average Pooling [43].

An activation layer performs a non-linear operation in a certain way depending on the type of activation function used. There are several activation functions like sigmoid, tanh, ReLU, and leaky ReLU. Figure 5 shows different activation functions used in deep learning.

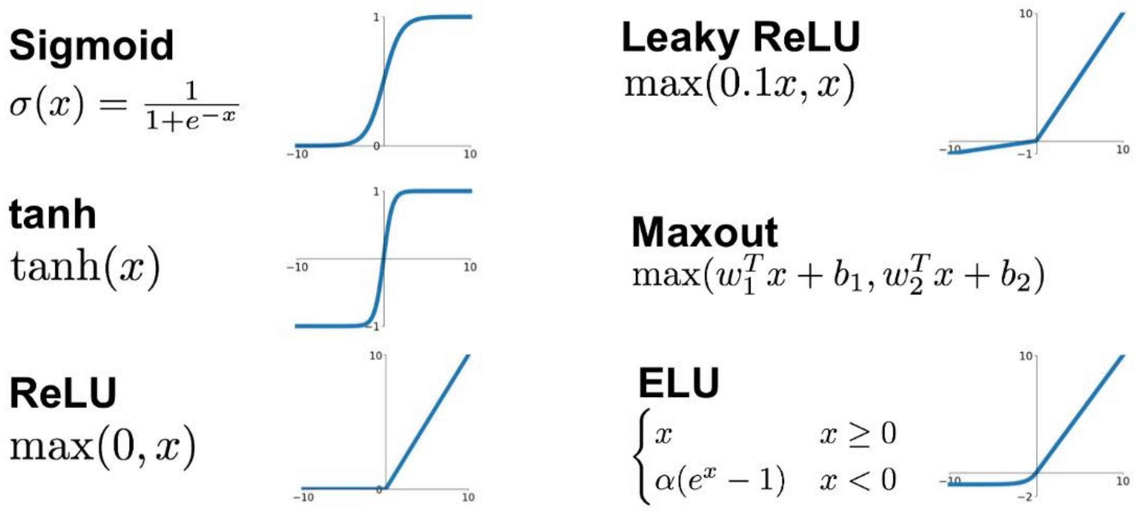


Figure 5 Different Activation Functions [44].

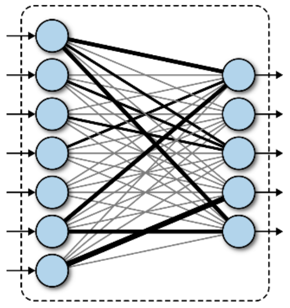


Figure 6 An example of Fully Connected Layer [45].

A fully connected layer is used to transform a high dimensional representation into a n-dimensional representation by connecting all the neurons in the input and fully connected layer. An example of fully connected layer is shown in Figure 6.

2.3 PointNet related architectures

A. PointNet:

PointNet [15] operates on the point clouds directly without converting them into other forms like voxels, a vector representation or rendering multiple views from point clouds. In order to account for the unordered nature of point clouds, PointNet proposed to use a symmetric function like maxpool to aggregate the features from the point clouds. PointNet consist of multi-layer perceptron (MLPs) to learn the point features and then apply symmetric function like maxpool to aggregate the point-wise features.

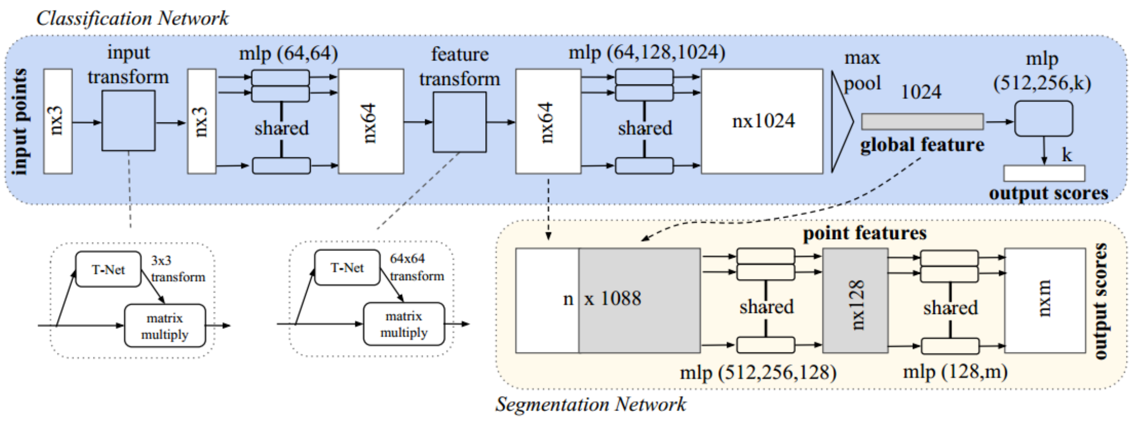


Figure 7 PointNet architecture [15].

PointNet consist of input transform and feature transform which are mini PointNet [15] like architectures. Input transform provides a 3×3 matrix which is applied on every point in the input point cloud and similarly feature transform provides a 64×64 matrix which is multiplied with pointwise features. A global feature vector is obtained by applying a symmetric maxpooling function such that this vector can be used in both a classification and segmentation network. The classification network outputs 'k' score and the segmentation network outputs '$n \times m$' scores. There are many PointNet inspired architectures. The architecture of [15] is shown in Figure 7.

B. Dynamic Graph CNN:

Inspired by [15]**,** [16] also operates on individual points to learn salient features. [15] considers each point individually and does not consider the relationships between point pairs. To account for this DGCNN constructs a *knn* graph and learns the local properties between the point pairs. DGCNN applies multi-layer perceptron on the *k* nearest graph constructed and applies symmetric maxpooling to get the aggregated features of the graph.

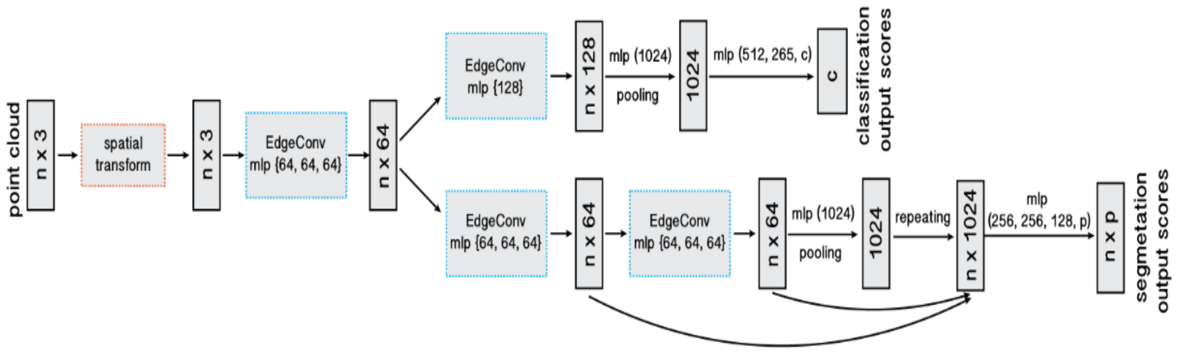


Figure 8 DGCNN architecture [16].

Figure 8 shows the DGCNN architecture, consisting of spatial transform and edge convolution block. The network consists of both a classification block and a segmentation block.

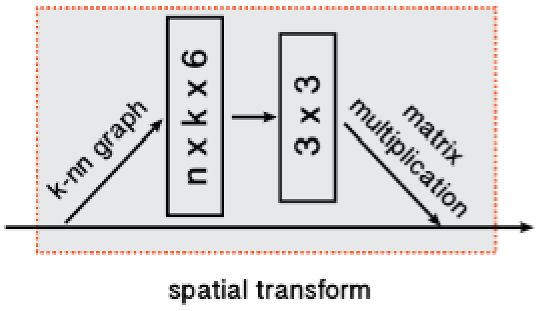


Figure 9 DGCNN spatial transform [16].

The spatial transform block aligns the input point cloud into a canonical space by applying a *3 × 3* matrix. This *3 × 3* matrix is estimated by constructing a *k*-nearest graph and then concatenating the point features with its *k*-nearest point features.

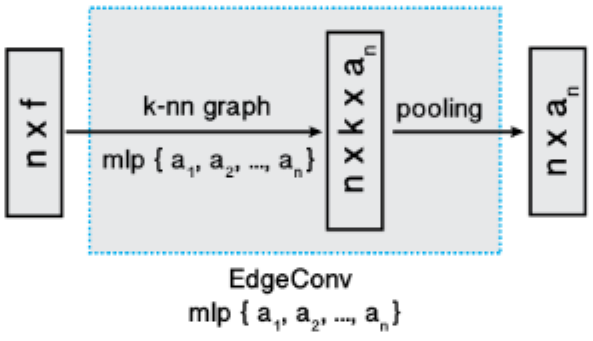


Figure 10 DGCNN EdgeConv [16].

The most important part of DGCNN is the edge convolution. Edge convolution is performed by constructing a *k*-nearest graph, then applying a multi-layer perceptron and then applying maxpooling to obtain the local features. The edge convolution operation is shown in (1).

$$e_{ijm} = ReLU\left(\Theta_m . (x_j - x_i) . (x_i)\right) \qquad (1)$$

In (1), x_i is each point in the input point cloud , x_j is each point in the *k*-nearest graph of x_i and Θ_m is the weight matrix which can be approximated by a multi-layer perceptron.

C. FoldingNet:

FoldingNet is a point cloud auto-encoder. FoldingNet [34] consists of a graph-based encoder on top of [15] and a folding-based decoder. FoldingNet [34] deforms a 2D grid into a 3D object surface of the point cloud.

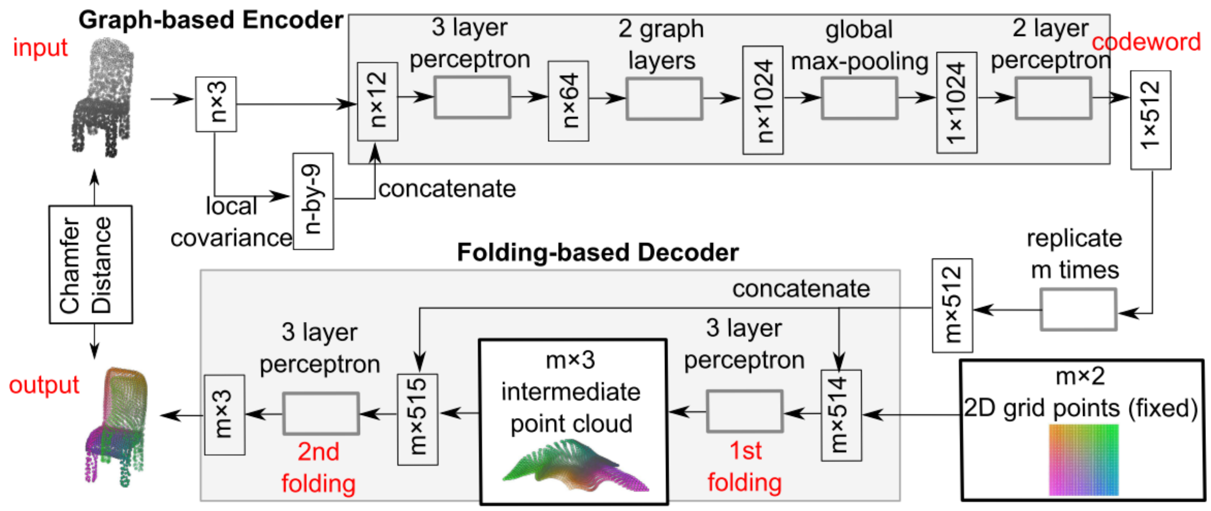


Figure 11 FoldingNet architecture [34].

The graph based encoder consists of multi-layer perceptron and graph based maxpooling layers. First a local covariance matrix of size 3×3 is computed using 3D positions of the points and its one hop neighbors. For every point in the input point cloud, the local covariance matrix is constructed to give a vector of $n \times 9$ and is concatenated with the input points of size $n \times 3$ to give a matrix of size $n \times 12$. The graph is a k-nearest graph computed by considering the k nearest neighbors of the 3D points and after that a maxpooling operation is performed to aggregate the features. A final codeword of size 1×512 is obtained as an output from the encoder.

The folding based decoder deforms the 2D fixed grid points to a 3D point cloud. The codeword from the graph-based encoder is fed into the decoder by replicating it m times and concatenated with the 2D grid points to obtain a matrix of size $m \times 514$. A multi-layer perceptron is applied on this matrix by processing it row-wise to obtain an intermediate point cloud. This concatenation and point-wise convolution is applied again on this intermediate point cloud to obtain a final output.

D. 3D Point Capsule Networks:

3D point capsule networks are an auto-encoder. They consist of dynamic routing between primary point capsules and latent capsules. Individual feature maps are computed by applying point-wise multi-layer perceptron. Primary point capsules are obtained by feeding these feature maps into multiple convolutional layers with different weights. Latent capsules are obtained by applying a dynamic routing algorithm to the primary point capsules. These latent capsules attend to different parts of the input point clouds because of the dynamic routing algorithm.

The decoder concatenates a 2D random grid of fixed size to the latent capsules obtained from the encoder. In order to reconstruct the point cloud of input size, latent capsules are replicated *m* times and then concatenated with a 2D random grid.

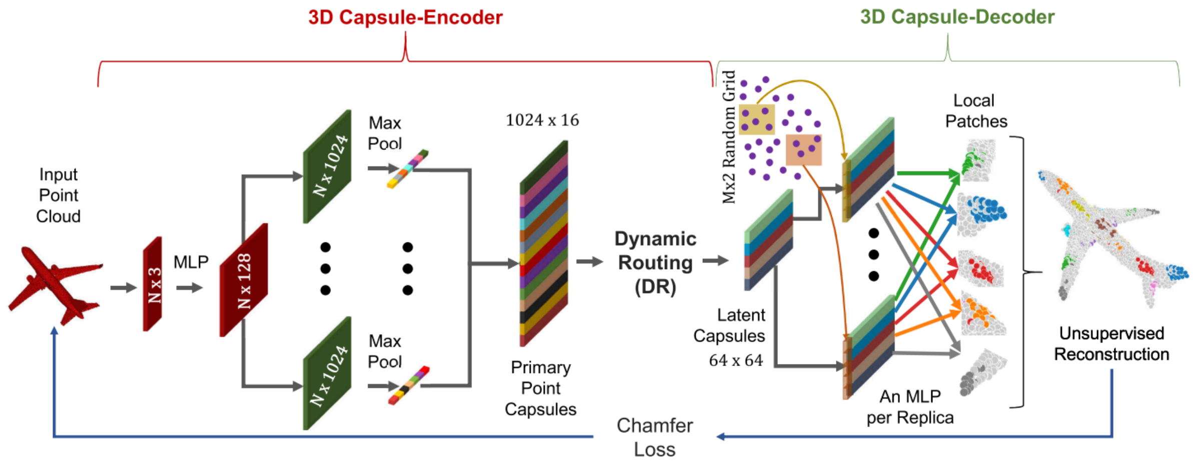
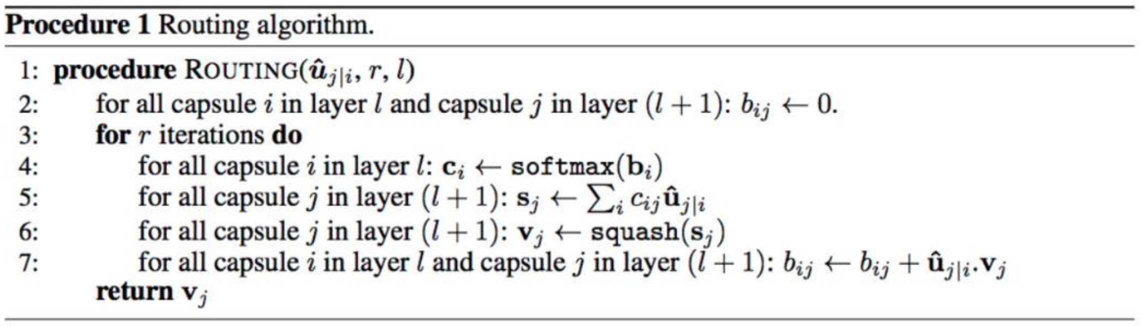


Figure 10 Architecture of 3D Point Capsule Networks [46].



Procedure 1 Routing algorithm.

```
1: procedure ROUTING($\hat{\mathbf{u}}_{j|i}$, $r$, $l$)
2:     for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3:     for $r$ iterations do
4:         for all capsule $i$ in layer $l$: $\mathbf{c}_i \leftarrow \mathtt{softmax}(\mathbf{b}_i)$
5:         for all capsule $j$ in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6:         for all capsule $j$ in layer $(l+1)$: $\mathbf{v}_j \leftarrow \mathtt{squash}(\mathbf{s}_j)$
7:         for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
       return $\mathbf{v}_j$
```

Figure 11 Dynamic Routing algorithm [47].

Dynamic routing algorithm is applied as shown in Figure 11. The algorithm is applied between all the point capsules in primary point capsules and all the capsules in latent capsules.

E. Point Completion Network:

Point completion network (PCN) is an encoder decoder architecture which completes an input partial point cloud and produces a complete point cloud. The encoder of [2] applies a point-wise multi-layer perceptron and then a symmetric function such as maxpooling to obtain a global feature vector. This global feature vector is replicated *m* times, concatenated with point-wise features and then applied multi-layer perceptron. A final global feature vector is computed by applying maxpooling. The final global vector is of size *1024*.

The decoder of PCN [2] takes global feature vector from the encoder as input and computes a coarse output by applying a multi-layer perceptron. The coarse output is concatenated with a 2D grid of fixed size with radius 4 and with the *m* times global feature vector. The shared multi-layer perceptron is applied to output a final detailed output.

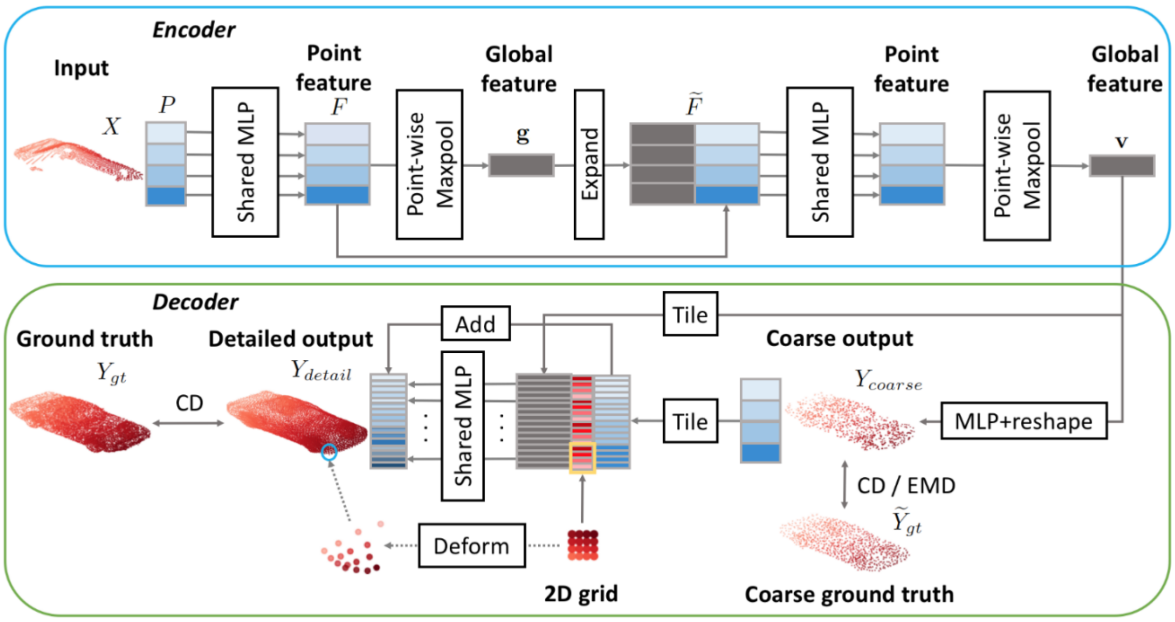


Figure 12 Architecture of PCN [2].

F. MeshNet:

MeshNet [48] is a neural network applied on 3D mesh data, considering each face of the 3D mesh as an operating unit. MeshNet [48] consist of a mesh convolution block, a structural descriptor and a spatial descriptor. The spatial descriptor takes centers as inputs and then applies a multi-layer perceptron to compute the center features. The structural descriptor takes neighboring indices, the normal of each face and the corners of each face as inputs, then computes features using face kernel correlation and face rotate convolution.

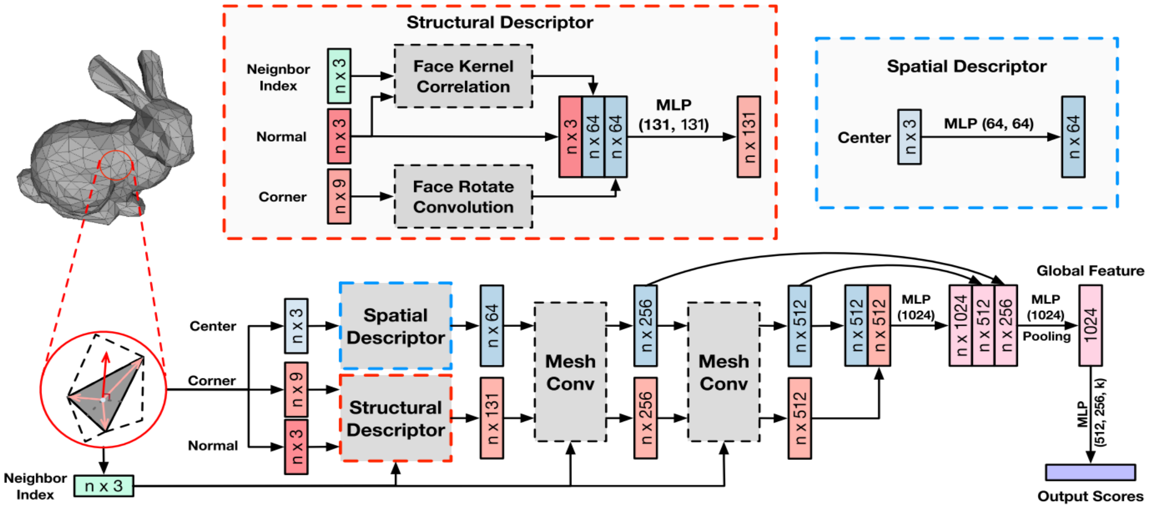


Figure 13 MeshNet architecture [48].

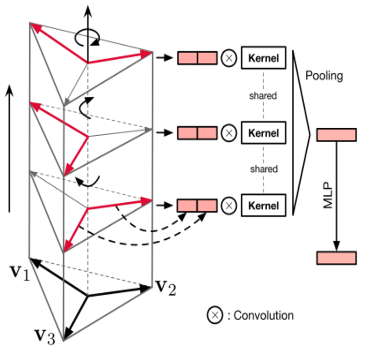


Figure 14 Face rotate convolution block [48].

The face rotate convolution block as shown in Figure 14 rotates the face and then applies a convolution operation on pairs of corner vectors. The face kernel correlation applies correlation between neighboring corners and a Gaussian kernel.

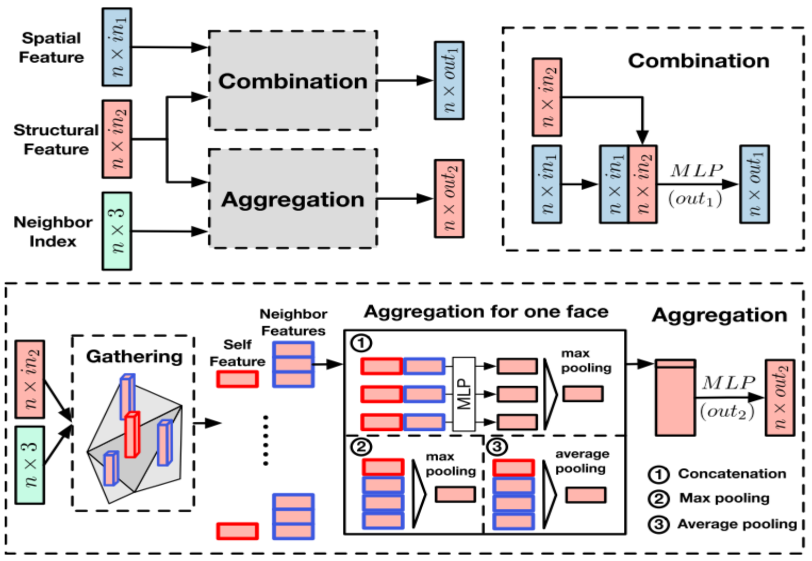


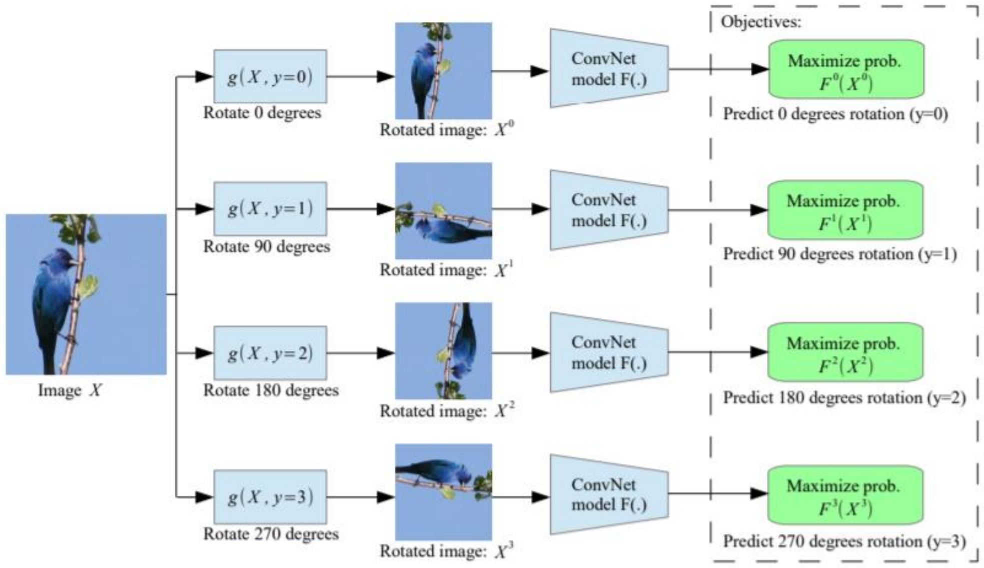
Figure 15 Mesh convolution block [48].

The mesh convolution block consists of a combination block and an aggregation block. The combination block combines the spatial features and structural features by concatenating spatial and structural features. The aggregation block takes neighboring indices and structural features as input, then computes the aggregated features by applying multi-layer perceptron and maxpooling operation.

2.4 Self-supervised Learning:

Transfer learning always helps the model if the task has less training data. But in many tasks, such as medical imaging and point completion, we cannot find a model to transfer the weights from. In tasks where transfer learning is not possible, self-supervised learning plays a key role in learning the input dataset features. In self-supervised learning we define a task known as a pretext task. We train our model and use these pretext task weights to perform transfer learning on our real, but limited training data. These pretext tasks don't need any ground truth data, rather the model defines a task and extracts its own ground truth for initial training of our model.

Different pretext tasks are defined in [58], [59], [60], [35]. [58] defined the task of colorization as the pretext task, [59] defined guessing the spatial information by randomly sampling the different patches of an image as the pretext task, [35] rotated the image into four different directions and defined classifying the angle rotation as the pretext task.



Objectives:
$g(X, y=0)$
Rotate 0 degrees
Rotated image: X^0
ConvNet model F(.)
Maximize prob. $F^0(X^0)$
Predict 0 degrees rotation (y=0)
$g(X, y=1)$
Rotate 90 degrees
Rotated image: X^1
Maximize prob. $F^1(X^1)$
Predict 90 degrees rotation (y=1)
Image X
$g(X, y=2)$
Rotate 180 degrees
Rotated image: X^2
Maximize prob. $F^2(X^2)$
Predict 180 degrees rotation (y=2)
$g(X, y=3)$
Rotate 270 degrees
Rotated image: X^3
Maximize prob. $F^3(X^3)$
Predict 270 degrees rotation (y=3)

Figure 16 An example of pretext task in self-supervised learning [35].

The pretext task used in [35] is shown in Figure 16, which shows the pretext task of rotating the image by different angles and then predicting the rotated class. Simple tasks like this helps the model to learn semantic features in a robust way. After pretext task learning, we use these weights to perform transfer learning on our ground truth labelled data. In the next chapter we will discuss the various loss functions we used in our models to optimize our tasks of point cloud completion and 3D mesh segmentation.

2.5 Loss Functions

A. Softmax and Cross Entropy Loss:

The softmax activation function takes an *N*-dimensional real vector as input and outputs an *N*-dimensional real vector with values in the range (0, 1) that add up to 1. It is usually used as the last layer before calculating loss.

$$S_j = \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \quad \forall j \in 1 \ldots N \tag{2}$$

Softmax uses (2) to calculate the probability of each class prediction in the input vector.

Cross entropy loss calculates the distance between the output of deep neural network and the original distribution. Cross entropy loss is calculated using (3).

$$H(y,p) = -\sum_{i} y_i \log p_i \tag{3}$$

B. Chamfer Distance Function:

$$CD(S_1,S_2) = \frac{1}{|S_1|} + \sum_{x \in S_1} \min_{y \in S_2} ||x-y||_2 + \frac{1}{|S_2|} + \sum_{y \in S_2} \min_{x \in S_1} ||y-x||_2 \tag{4}$$

The Chamfer Distance (CD) as shown in (4) calculates the average distance between the input point cloud and output point cloud. In (4) S_1 denotes the predicted complete point cloud and S_2 denotes the ground truth complete point cloud.

C. Earth Mover Distance (EMD) Loss Function*:*

$$EMD(S_1,S_2) = \min_{\phi: S_1 \to S_2} \frac{1}{|S_1|} + \sum_{x \in S_1} ||x - \phi(x)||_2 \tag{5}$$

The Earth Move Distance (EMD) as shown in (5) finds a bijection from the input point cloud to the output point cloud, minimizing the average distance between corresponding points. Here bijection function finds the minimum cost to move from the predicted point coordinates to the actual ground truth coordinates. In the next chapter, we will discuss the various models and the training strategies used to train these models in great detail.

Chapter 3

Methodology

In this chapter, we will explore all the different architectures we experimented with for the task of point cloud completion and 3D mesh segmentation. Each architecture is shown and discussed in great detail. A training strategy for each architecture is also provided. For the point cloud completion, each architecture consists of an encoder and a decoder, the output of the encoder is fed into the decoder to get the complete point cloud. Several approaches of point cloud feature extractions like PointNet, Graph based, and Capsule based are discussed. The application of self-supervised in the context of point cloud completion is also discussed. For the 3D mesh segmentation, we have experimented with different modified PointNet based and Mesh-based architectures.

3.1 Multistage point completion network

A. Encoder:

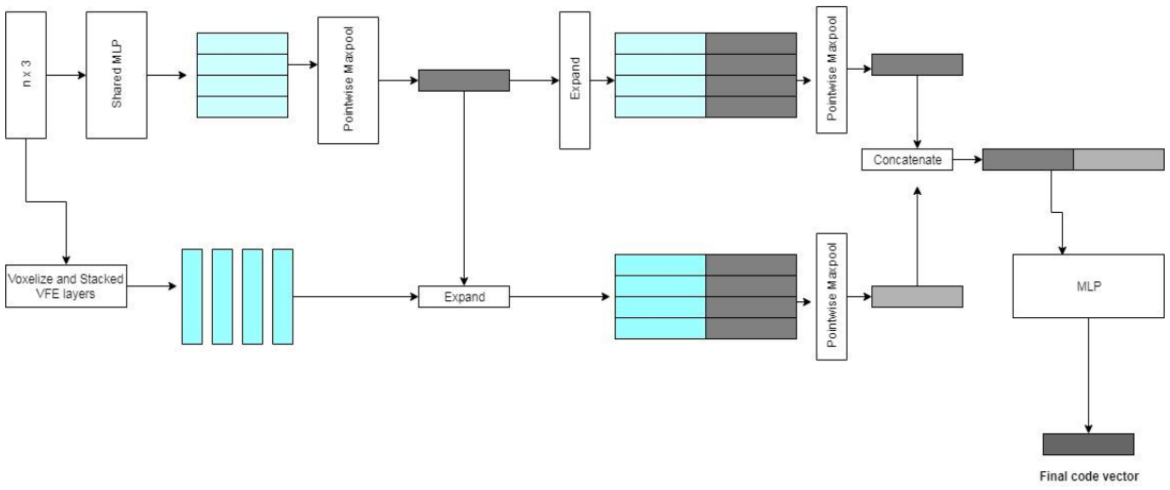


Figure 17 Encoder for multistage point completion network.

Figure 17 shows the proposed encoder for point cloud completion. The Point Completion Network (PCN) [2] extracts the point-wise features using style shared multi-layer perceptron, concatenates point-wise features with global features and then applies shared multi-layer perceptron layers.

Many 3D object detection networks [36,55] voxelize the input point cloud and then performs multi-layer perceptron. Since our input is a partial point cloud, interacting different voxels and global features is desirable to extract the features in a better way. Input point clouds are voxelized and voxel features are extracted using voxel feature extraction (VFE) layers.

B. Voxel Feature Extraction Layer:

Voxelization is converting the input point cloud into a 3D grid, whereby each sub-grid is called a voxel. Each voxel is assigned fixed number of points belonging to that voxel. If a voxel doesn't contain minimum of number of points padding is used. VFE layers are used to extract the voxel-wise features.

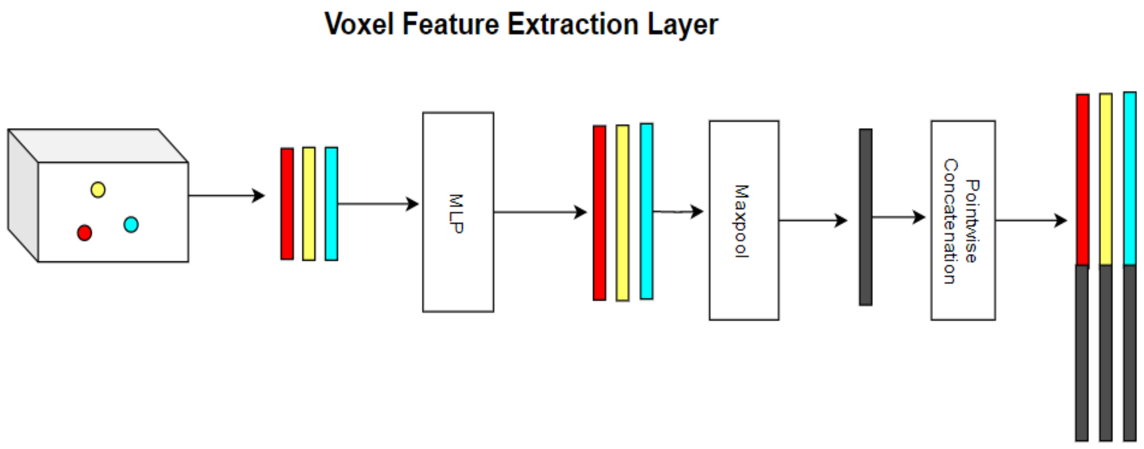


Figure 18 Voxel feature extraction Layer.

After extracting voxel-wise features, these voxel-wise features are concatenated with global features extracted by a shared multi-layer perceptron. This ensures learning the relationships between different points in the input point cloud. Without voxelization, it is difficult to learn the relationships in the input partial point cloud. After applying a shared multi-layer perceptron to the concatenated voxel feature vector and global feature vector, a symmetric maxpooling operation is applied. This vector is concatenated with the other global feature vector and then passed through a shared multi-layer perceptron. A final vector of size *1024* is obtained as an output from the encoder. An example of voxel feature extraction layer is shown in Figure 18.

C. *Decoder*:

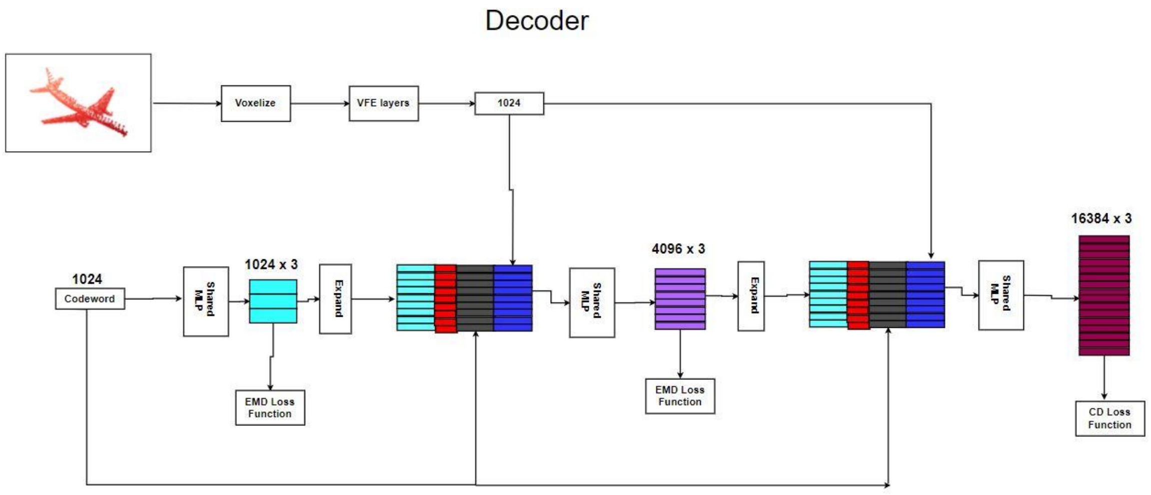


Figure 19 Decoder for multistage point completion network.

Our proposed decoder is shown in Figure 19. PCN [2] uses the codeword from the encoder as input and optimizes the codeword in the form of a coarse and fine output. The decoder shown in Figure 19 optimizes the code in 3 stages- in the form of a coarse output, middle output and final output. Since the problem is completing the input partial point cloud, a codeword from the encoder will have global information. To use this information effectively, the model needs to learn how to optimize the different sizes of point clouds. The proposed decoder outputs a coarse point cloud, a middle point cloud and a fine complete point cloud.

As shown in Figure 19, the shared multi-layer perceptron is applied to the codeword obtained from the encoder to output a coarse output of size 1024×3. FoldingNet [34], AtlasNet [54] and PCN [2] deform a 2D grid to the 3D point cloud. Our decoder deforms the 2D grid in multiple stages. The coarse output is concatenated with a tiled fixed 2D grid and tiled codeword to produce a middle output of size 4096×3. The middle output is concatenated with another tiled fixed 2D grid and a tiled codeword to output a final complete point cloud. The coarse output and middle output are optimized using the EMD (4) loss function, while the final output is optimized using the CD (3) loss function.

D. Training Strategy

The encoder and decoder are coded in the TensorFlow framework. The model is trained for *15* epochs with an initial learning rate of *0.0001* and uses exponential decay with decay rate *0.1*. The learning rate is decayed at several steps during training. The model uses batch normalization [53] at each multi-layer perceptron layer. Training was performed in two different ways. In the first way the model is trained end to end, optimizing the losses of coarse output, middle output and complete point clouds output. During initial iterations of training the middle output and complete output losses are not as important as the initial coarse output. This is done using two parameters, α and β. Starting from value *0.01*, both parameters values increased after every '*m*' iterations.

In the second way of training, the model is divided into three stages. The stage 1 model is trained only for the coarse output. This coarse output is taken by a stage 2 model and trained for middle output. The stage 3 model takes middle output as input and trained only for complete point cloud. Figure 20 shows the stage-wise learning procedure.

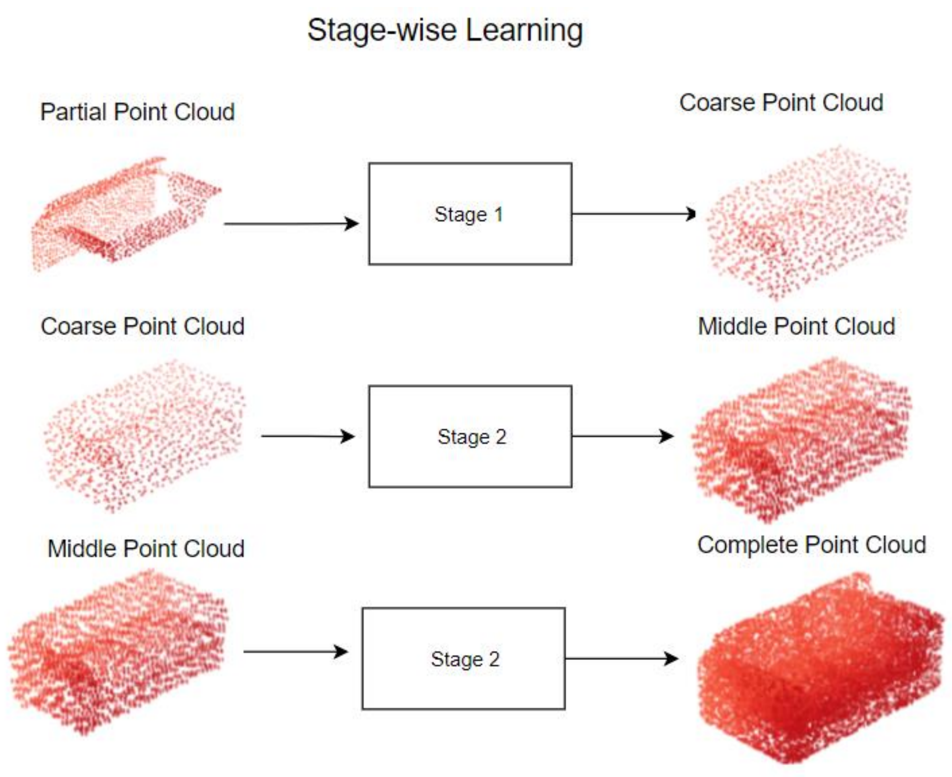


Figure 20 Stage-wise Learning.

3.2 Point Completion Network using Edge Convolution:

A. Edge convolution based network:

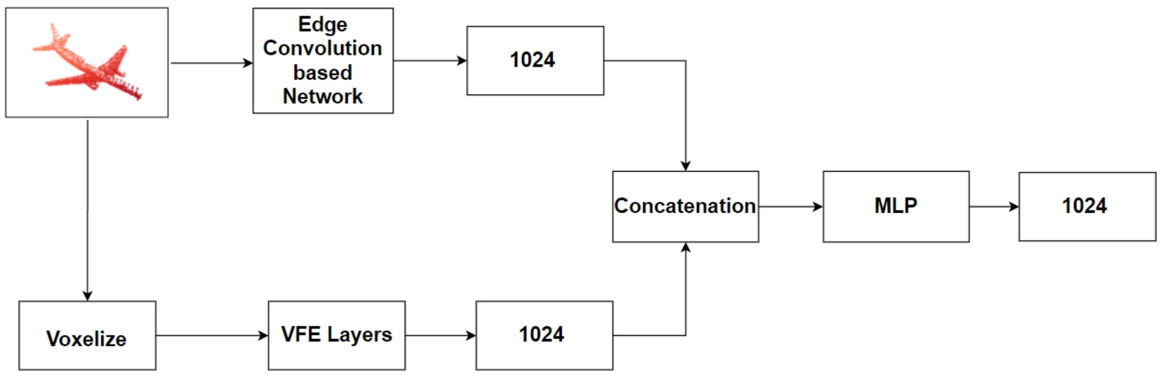


Figure 21 Edge convolution based encoder architecture.

DGCNN [16] employs edge convolution to effectively extract the local features by constructing a *k*-nearest graph, applies multilayer perceptron and then maxpooling operation. PCN [2] extracts the pointwise features by using PointNet [15] style architecture. The architecture of the edge convolution based point completion network is shown in Figure 21. The input partial point cloud is voxelized, voxel features are extracted using VFE (Voxel Feature Extraction) layers, concatenated with the final vector obtained from the edge convolution network, and then a multiplayer perceptron is applied to obtain the final code vector.

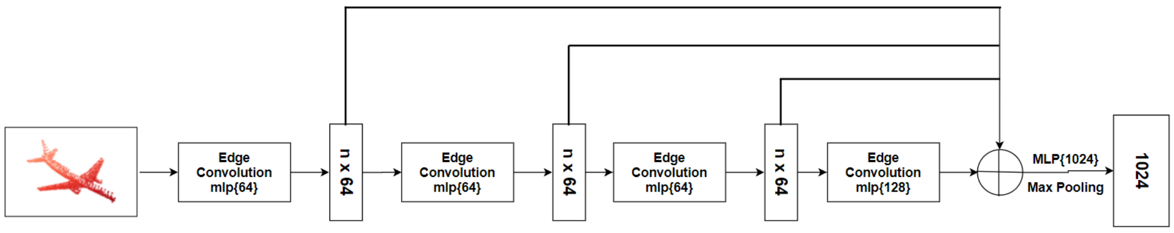


Figure 22 Complete architecture of edge convolution network.

The detailed architecture of the edge convolution network is shown in Figure 22. The architecture contains *4* edge convolution blocks, the output of the edge convolution blocks are concatenated, applied multilayer perceptron and then maxpooling to obtain a *1024* vector. The input to the network is a partial point cloud of shape 2048×3. The operation of the edge convolution is same as in [16]. The input to the VFE layers is of shape $m \times p \times 3$, where m is the number of voxels, p is the maximum number of points in the voxel. Interacting the points in different voxels in necessary to learn the information of missing points in the input point cloud. Because of this reason, we employed 2D convolutions to extract the features from the voxels. The final vector from the VFE layers is of shape *1024*.

B. Decoder:

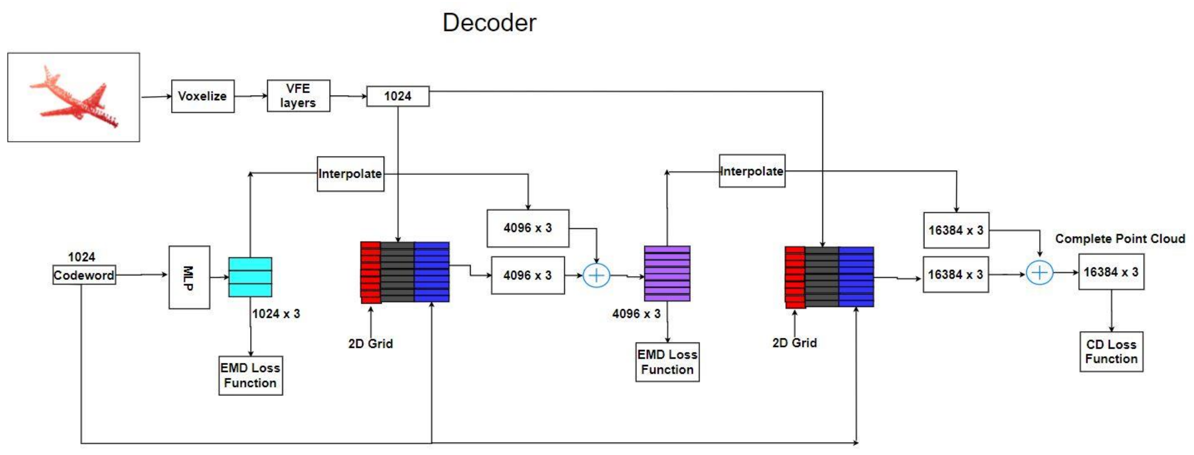


Figure 23 Architecture of the decoder using interpolation.

The modified decoder which employs an interpolation technique, instead of tiling, is shown in Figure 23. The decoder optimizes the complete point cloud in multiple stages: the coarse stage, the middle stage and the complete stage. The ground truths for each of the stages are generated by using FPS (Farthest Point Sampling) which is used in [14]. The FPS algorithm has been shown to be advantageous as compared to random sampling in sampling the point clouds. Both, the middle output and the complete point cloud are generated by concatenating a 2D grid of fixed size to the code word from the encoder and the final vector from the VFE layers. The coarse and the middle point are optimized using EMD loss function, and the complete point cloud is optimized using CD loss function.

In [2], the tiled coarse point cloud is concatenated with the 2D random grid to generate the complete point cloud. A better way to use the information of the coarse point cloud is an interpolation, as during tiling, duplication of the same features adds minimal information for the generation of the complete point cloud. The decoder used in this architecture interpolates the coarse point cloud to the shape of the middle point cloud and then added to the middle point cloud features generated using the folding technique. Similarly, the middle point cloud is interpolated to the shape of the complete point cloud and then added to the complete point cloud generated using the folding technique.

$$f^j(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i{}^j}{\sum_{i=1}^{k} w_i(x)} \tag{6}$$

$$w_i(x) = \frac{1}{d(x, x_i)} \qquad (7)$$

The interpolation operation is shown in (6), where $f^j(x)$ is the interpolated feature from the lower level (coarse point cloud or middle point cloud). ${f_i}^j$ are features of the k-nearest neighbors of the higher level (middle point cloud or complete point cloud) in the lower level. Each neighbor is multiplied with a weight value which is equal to the distance of the point in a higher level to the lower level. After generating the middle point cloud using the folding technique, we interpolate the coarse point cloud using the middle point cloud and add them together. Similarly, after generating the complete point cloud, we interpolate the middle point cloud using the complete point cloud and add them together. The number of nearest neighbors used to interpolate has been empirically solved to be *3*. The interpolation technique shown in (6) is proposed in PointNet++ [14]. In [14], it is used for segmentation of the input point cloud.

3.3 Capsule Point Completion Network

A. *Capsule based Decoder:*

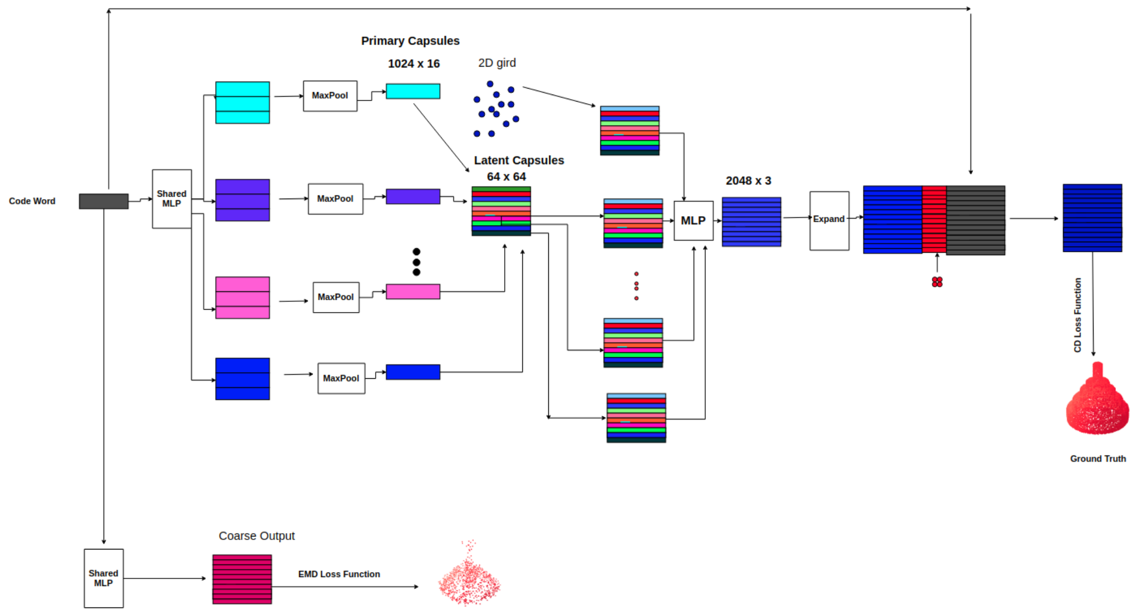


Figure 24 Capsule based decoder architecture.

Dynamic routing between capsules [47] has been shown to have effective performance on the MNIST dataset with very shallow architectures. [52] extended the dynamic routing architecture for multiple layers. [46] is the first architecture to apply the concept of dynamic routing to 3D point clouds. They have shown impressive results with reconstructing the input point cloud. Our capsule-based architecture as shown in Figure 21 uses the dynamic routing algorithm between a primary point capsule and latent capsules to obtain the final complete point cloud.

The primary point capsules are produced by applying a different shared multi-layer perceptron and then maxpooling operation to the codeword obtained from the encoder. To compute the latent capsules, the dynamic routing algorithm is applied to the primary point capsules. Applying the dynamic routing algorithm makes the model attend to different parts of the coarse output. The latent capsules are concatenated with the tiled fixed 2D grid and then applied shared multi-layer perceptron to compute the final complete point cloud. The model is optimized in stages of coarse output and final complete point cloud.

$$V_j = \frac{||\, S_j||^2}{1 + ||\, S_j||^2} \; \frac{S_j}{||\; S_j\; ||} \tag{7}$$

In (7) V_j is the vector output of capsule j and S_j is its total input.

$$S_j = \sum_i c_{ij}\, U_{ij} \tag{8}$$

In (8) j denotes the latent capsule and i denotes primary capsule. Input to a latent capsule is the prediction vectors of primary capsule multiplied by coupling coefficient c_{ij}.

$$U_{ij} = W_{ij} u_i \tag{9}$$

Equation (9) shows the prediction vector of a primary capsule is obtained by multiplying output of primary capsule u_i with the weight matrix.

$$c_{ij} = \exp(b_{ij}) \Big/ \textstyle\sum_k \exp(b_{ik}) \tag{10}$$

Equation (10) shows how coupling coefficients are computed. The coupling coefficients from primary capsule i to latent capsule j sum to 1 and are computed by (10) which is called routing softmax, whose initial logits b_{ij} are the log prior probabilities that primary capsule i connected to upper latent capsule j.

B. Training Strategy

The model is coded in the TensorFlow framework and trained for *10* epochs with an initial learning rate of *0.001*. The learning rate is decayed after every *1000* iterations with decay rate of *0.1*. For the coarse output loss, the EMD [23] loss function is used and for the complete point cloud, the CD [23] loss function is used. The number of iterations for calculating coupling coefficients in each training iteration are three.

3.4 Multiview Point Completion Network

A. Multiview Encoder:

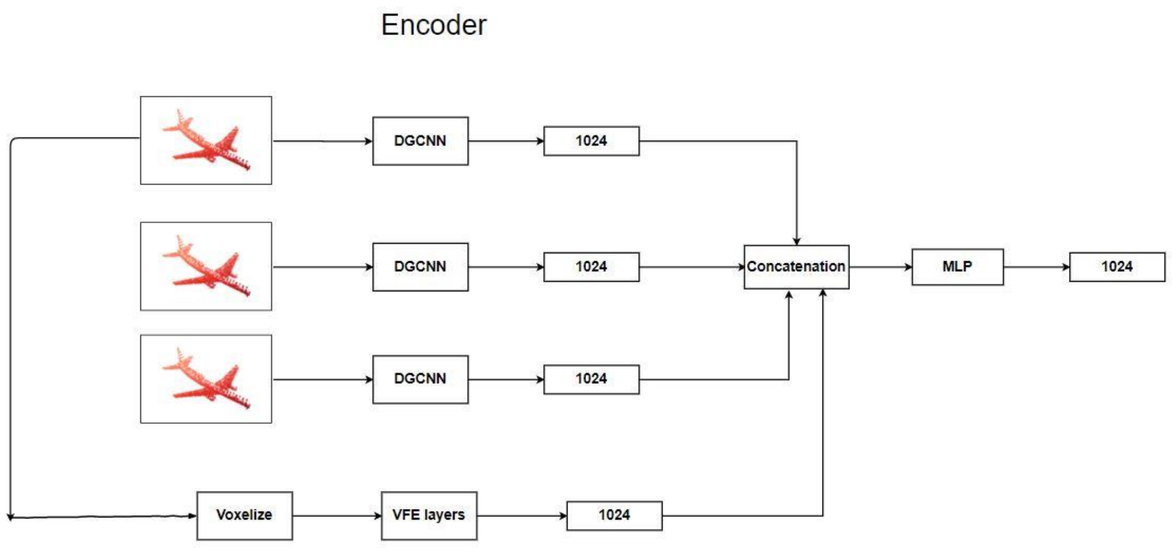


Figure 25 Multiview encoder.

The encoder of the multiview point completion network using DGCNN [16] is shown in Figure 22. The input partial point cloud is rotated by 1 degree and 3 degrees. All three point clouds, the input point cloud without rotation and the two rotated point clouds are fed into DGCNN [16] to compute the global vector of size *1024*. To construct the k-nearest neighbors in DGCNN [16], we used *k*=20 and computed the edge features as in [16]. The input partial point cloud is voxelized and computed voxel-wise features using voxel feature extraction (VFE) layers, and applied maxpooling to compute the global vector of size *1024*. We used three stacked VFE layers to extract the voxel-wise features. In general, VFE layer computes point-wise features and then applies symmetric maxpooling function to obtain the global voxel-wise features. In this way, we applied 3 VFE layers after converting the input partial point cloud to voxels. These global voxel-wise features are concatenated with the point-wise features to obtain the final voxel-wise features. All the global vectors computed from the rotated point clouds and VFE layers are fed into a multi-layer perceptron to compute the final codeword of size *1024*. The multi-layer perceptron architecture has *512 – 1024* nodes to compute the final codeword.

B. Decoder

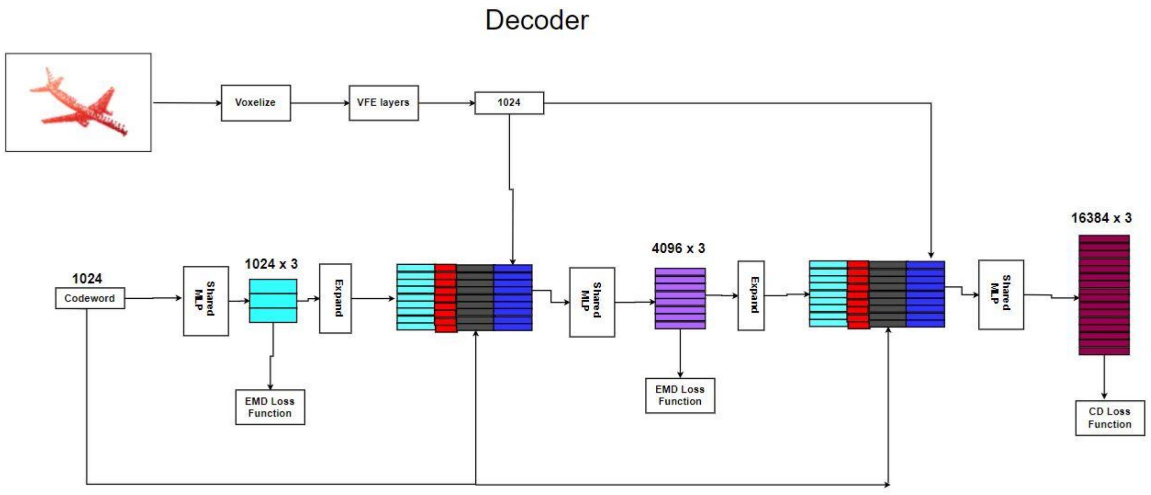


Figure 26 Decoder for Multiview point completion network.

The decoder for the multiview point completion network is shown in Figure 23. The decoder receives a codeword and a global voxel-wise feature vector from the encoder and optimizes it in multiple stages. In the first stage, the codeword is converted into a coarse point cloud of shape *1024 × 3* by applying a shared multi-layer perceptron. The loss is calculated using the EMD [23] loss function. In the second stage, a 2D random grid is concatenated to the coarse vector obtained in the first stage. The codeword is tiled and concatenated and global voxel-wise feature is concatenated to obtain a middle vector of shape *4096 × 2053*. The shared multi-layer perceptron is applied to this vector and the loss is calculated by comparing with the ground truth using EMD [23] loss function. In the third stage, the process is repeated as in the second stage, whereby a 2D random grid is concatenated to the middle vector, a coarse vector obtained in the first stage is tiled and concatenated, and the codeword from the encoder is tiled and concatenated with a voxel-wise global feature vector. Then shared multi-layer perceptron is applied to compute the final complete point cloud.

The ground truth for the coarse point cloud and the middle point cloud is generated by using the farthest sampling algorithm [14] from the complete point cloud ground truth. Iterative farthest sampling is used since it is better than random sampling. The coarse point cloud and the middle point cloud is optimized using the EMD [23] loss function, as the cost of EMD [23] is higher than of the CD [23] loss function.

C. *Training Strategy*

The model is coded in the TensorFlow framework and trained for *10* epochs with an initial learning rate of *0.001*. The learning rate is decayed after every 1000 iterations with a decay rate of *0.1*. For the coarse output loss, the EMD [23] loss function is used, and for the complete point cloud, the CD [23] loss function is used.

3.5 Self-supervised based point completion network

A. *Self-supervised Encoder:*

Encoder

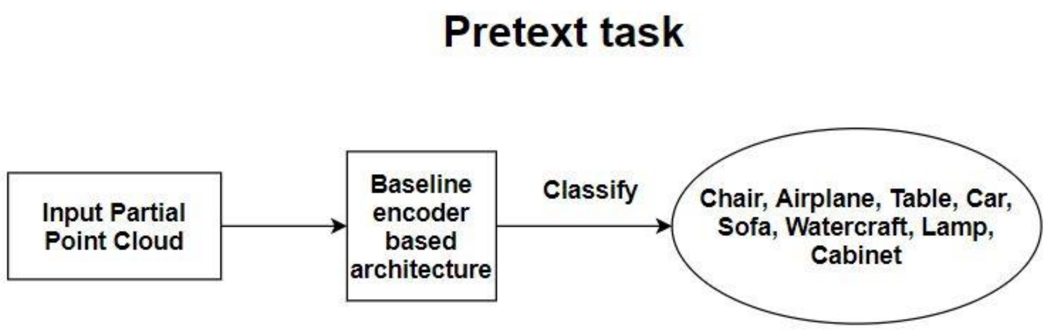

Pretext task
Input Partial Point Cloud
Baseline encoder based architecture
Classify
Chair, Airplane, Table, Car, Sofa, Watercraft, Lamp, Cabinet

Figure 27 Self-supervised pretext task for point completion network.

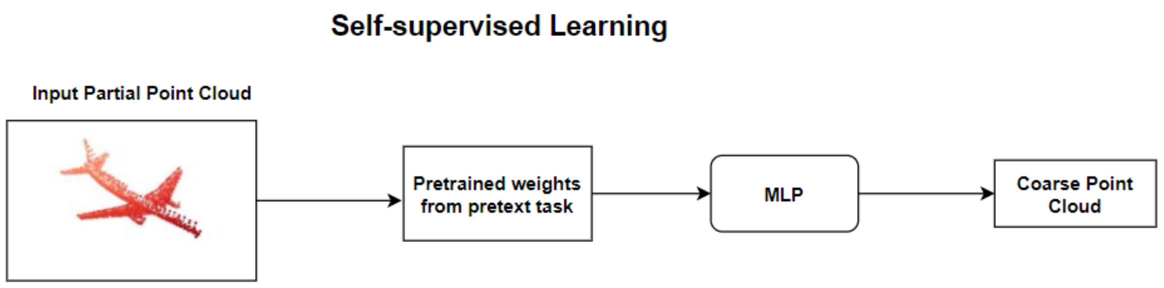

Self-supervised Learning
Input Partial Point Cloud
Pretrained weights from pretext task
MLP
Coarse Point Cloud

Figure 28 Self-supervised learning for optimizing coarse point cloud.

The pretext task for the self-supervised point completion network is shown in Figure 27. The pretext we have chosen is the classification task. We used the same architecture as the PCN [2] model to extract the input partial point cloud features and use these features to classify the input partial point clouds. The total number of classification categories are eight. We trained the encoder for this classification task. After training the model for this rotation pretext task, we perform transfer learning using the pretext weights to compute the global vector of size *1024*. The model trained on the pretext task learns the semantic features of the input partial point clouds in an improved fashion. Using transfer learning with those weights extracts the features of the input partial clouds with better results. To classify the input partial point clouds, the model must identify certain features which helps in correctly classifying the point clouds belonging to different categories. This identification of features by the model for this pretext task helps the task of point completion with less data.

B. Training Strategy

The model is coded in the TensorFlow framework and trained for *10* epochs with an initial learning rate of *0.001*. The learning rate is decayed after every 1000 iterations with decay rate of *0.1*. For the coarse output loss, the EMD [23] loss function is used and for the complete point cloud, the CD [23] loss function is used. We have used 40% less data for this model compared to the other models to optimize the coarse point clouds.

3.6 Mesh Segmentation

For the robot gripper like a suction cup to hold the object firmly, it has to identify the good faces on the surface of the object. Since 3D data can be represented in different ways, representing the 3D object in the form of 3D mesh provides more information about the connectivity of faces on the surface of the object. We represented 3D objects in the form of a 3D mesh and apply deep learning based architectures to identify the good faces on the object. This problem is formulated as mesh segmentation, where each face in the 3D mesh can be a good or bad face.

A. PointNet Center:

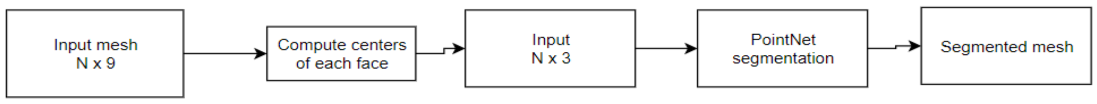


Figure 29 PointNet Center.

The PointNet center [15] architecture is shown in Figure 27. Input to the architecture is a 3D mesh. The mesh consists of faces and vertices. The faces contain information of connection between vertices and vertices are actual 3D points of the mesh. We computed the centers of each face in the 3D input mesh of shape $N \times 9$ to convert into shape of $N \times 3$. Then PointNet [15] segmentation architecture is applied to this modified input to output segmented mesh.

B. PointNet Mesh:

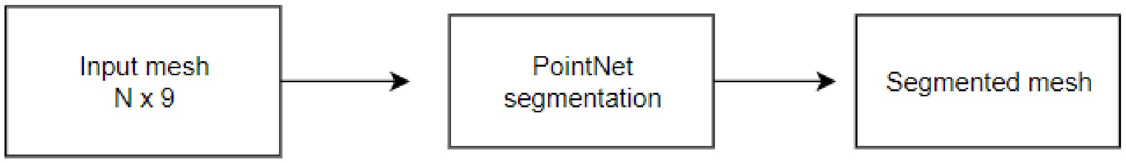


Figure 30 PointNet Mesh.

The PointNet mesh architecture is shown in Figure 28. PointNet [15] uses the 3D point clouds of shape N × 3 as inputs, whereas the architecture shown in Figure 28, uses 3D mesh as input. We concatenated the vertices of each face in the 3D mesh to convert into a vector of shape $N \times 9$. Then segmentation architecture in [15] is applied to segment the input mesh.

C. DGCNN Center:

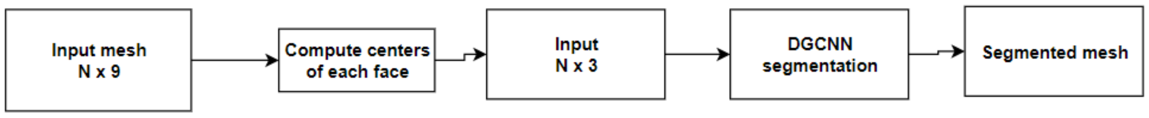


Figure 31 DGCNN Center.

DGCNN [16] architecture we used for mesh segmentation is shown in Figure 29. As shown in Figure 29, we computed the centers of each face in the input 3D mesh of shape $N \times 9$ and converted into input of shape $N \times 3$. The DGCNN [16] architecture is applied to segment the 3D input mesh. The number of segmentation classes in the output is two, one is good face and the other is bad face. We used the cross entropy loss in the output layer to calculate the loss between the predicted and ground truth.

D. MeshNet

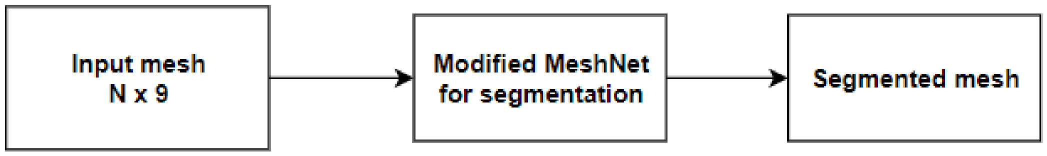


Figure 32 MeshNet for segmentation.

MeshNet [48] considers the face information, normal vectors of each face, and neighboring corners of each face and centers of each face to perform the classification task. We processed the input 3D mesh in the same way as in [48], computed neighboring corners of each face, normal vectors of each face and centers of each face. We modified [48] for segmentation and used same architecture for the input 3D mesh. In the next chapter, we will discuss the datasets used in training our different architectures.

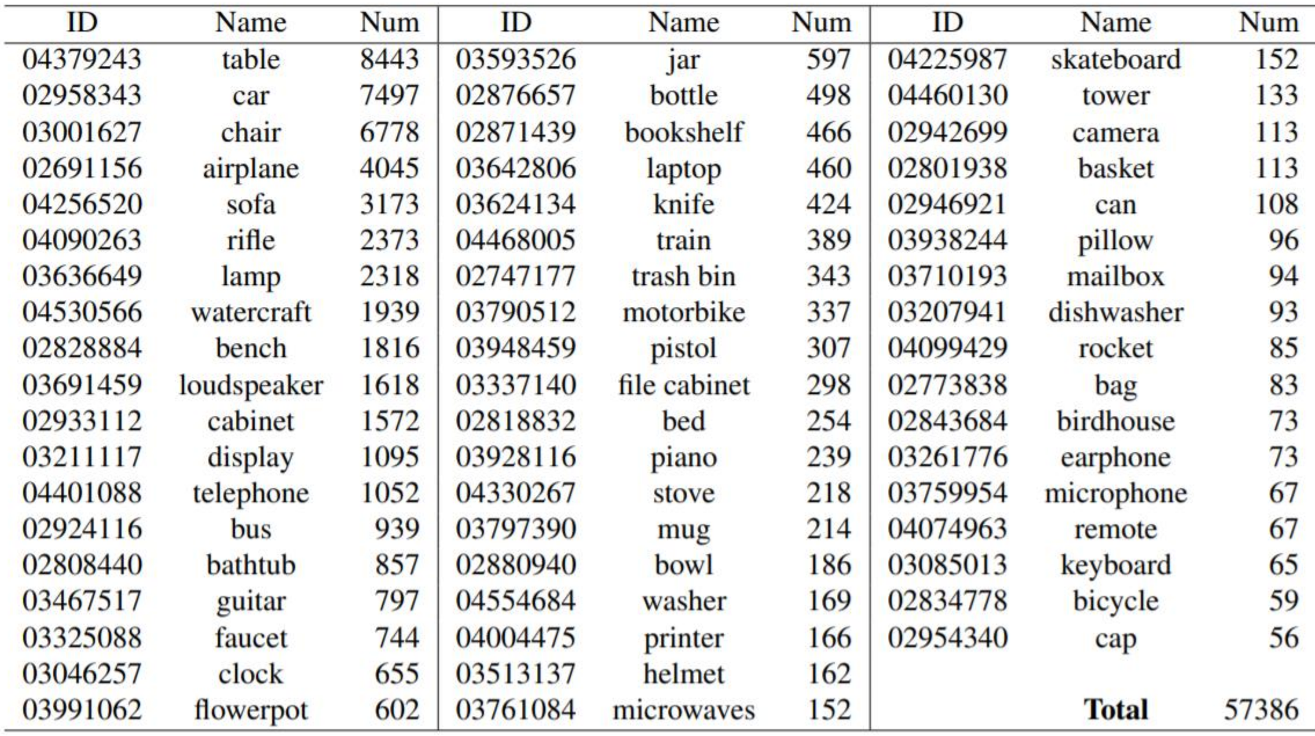
Chapter 4

Datasets

4.1 Datasets

A. ShapeNet

ShapeNet[8] is a large-scale annotated repository of 3D CAD models of different objects. It provides a rich set of annotations for every object. ShapeNet provides variety of 3D models to evaluate the performance of 3D shape reconstruction, 3D part segmentation and 3D object classification. There are total *55* common categories of 3D objects in ShapeNet.



ID	Name	Num	ID	Name	Num	ID	Name	Num
04379243	table	8443	03593526	jar	597	04225987	skateboard	152
02958343	car	7497	02876657	bottle	498	04460130	tower	133
03001627	chair	6778	02871439	bookshelf	466	02942699	camera	113
02691156	airplane	4045	03642806	laptop	460	02801938	basket	113
04256520	sofa	3173	03624134	knife	424	02946921	can	108
04090263	rifle	2373	04468005	train	389	03938244	pillow	96
03636649	lamp	2318	02747177	trash bin	343	03710193	mailbox	94
04530566	watercraft	1939	03790512	motorbike	337	03207941	dishwasher	93
02828884	bench	1816	03948459	pistol	307	04099429	rocket	85
03691459	loudspeaker	1618	03337140	file cabinet	298	02773838	bag	83
02933112	cabinet	1572	02818832	bed	254	02843684	birdhouse	73
03211117	display	1095	03928116	piano	239	03261776	earphone	73
04401088	telephone	1052	04330267	stove	218	03759954	microphone	67
02924116	bus	939	03797390	mug	214	04074963	remote	67
02808440	bathtub	857	02880940	bowl	186	03085013	keyboard	65
03467517	guitar	797	04554684	washer	169	02834778	bicycle	59
03325088	faucet	744	04004475	printer	166	02954340	cap	56
03046257	clock	655	03513137	helmet	162			
03991062	flowerpot	602	03761084	microwaves	152		**Total**	57386

Figure 33 Different categories for ShapeNet dataset [8].

Figure 31 shows the different categories and their corresponding dataset size in ShapeNet.

Figure 34 Examples of ShapeNet dataset [8].

B. Point Completion Networks:

For the point completion task, we need a dataset containing pairs of partial and complete points clouds. We used the same method as in PCN [2] to generate complete and partial point clouds. The complete point clouds are generated by sampling 16,384 points from mesh surfaces of 3D CAD models of the ShapeNet dataset. To generate partial point clouds, we back projected 2.5D depth images into 3D. There are total of eight categories of objects chosen for this task: airplane, cabinet, chair, car, lamp, sofa, table, vessel. Back-projecting the depth images into 3D brings the data distribution to the real-world sensor data. Since real world data won't contain detailed information of 3D objects, synthetic 3D objects like those in ShapeNet provide rich 3D information to process.

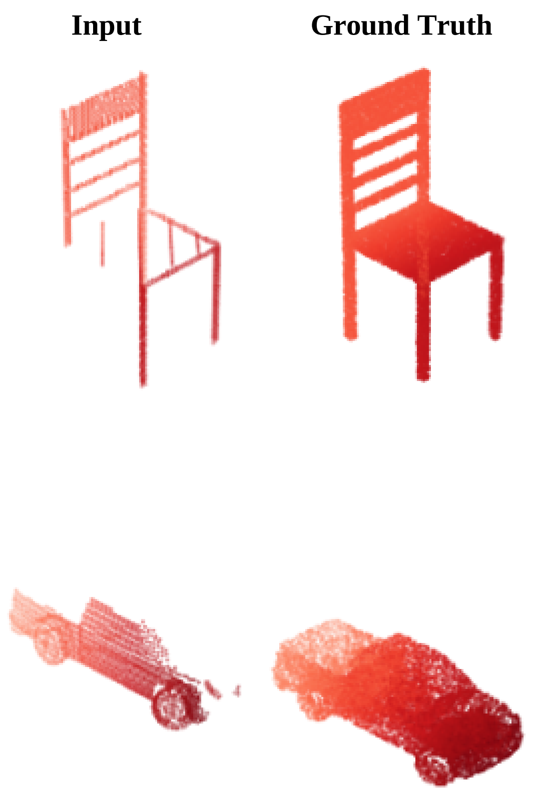


Figure 35 Examples of PCN dataset [2].

Examples of dataset containing input partial point clouds and ground truth complete point clouds are shown in Figure 35.

C. *Phenix Automation:*

For the robot grippers like suction cups to hold the objects firmly, the 3D mesh object should contain a minimum flat surface area. Since a gripper can hold the object anywhere, a segmentation of the 3D mesh satisfying the minimum area requirements should be done. Present datasets like ShapeNet [8], ScanNet [49], S3DIS [50] do not contain 3D objects which would typically might be found in industrial machinery environments. As such, we have created our own dataset grabbing different 3D CAD models from GrabCAD [51] and then manually annotated the dataset. For annotating the dataset, we developed a MATLAB tool.

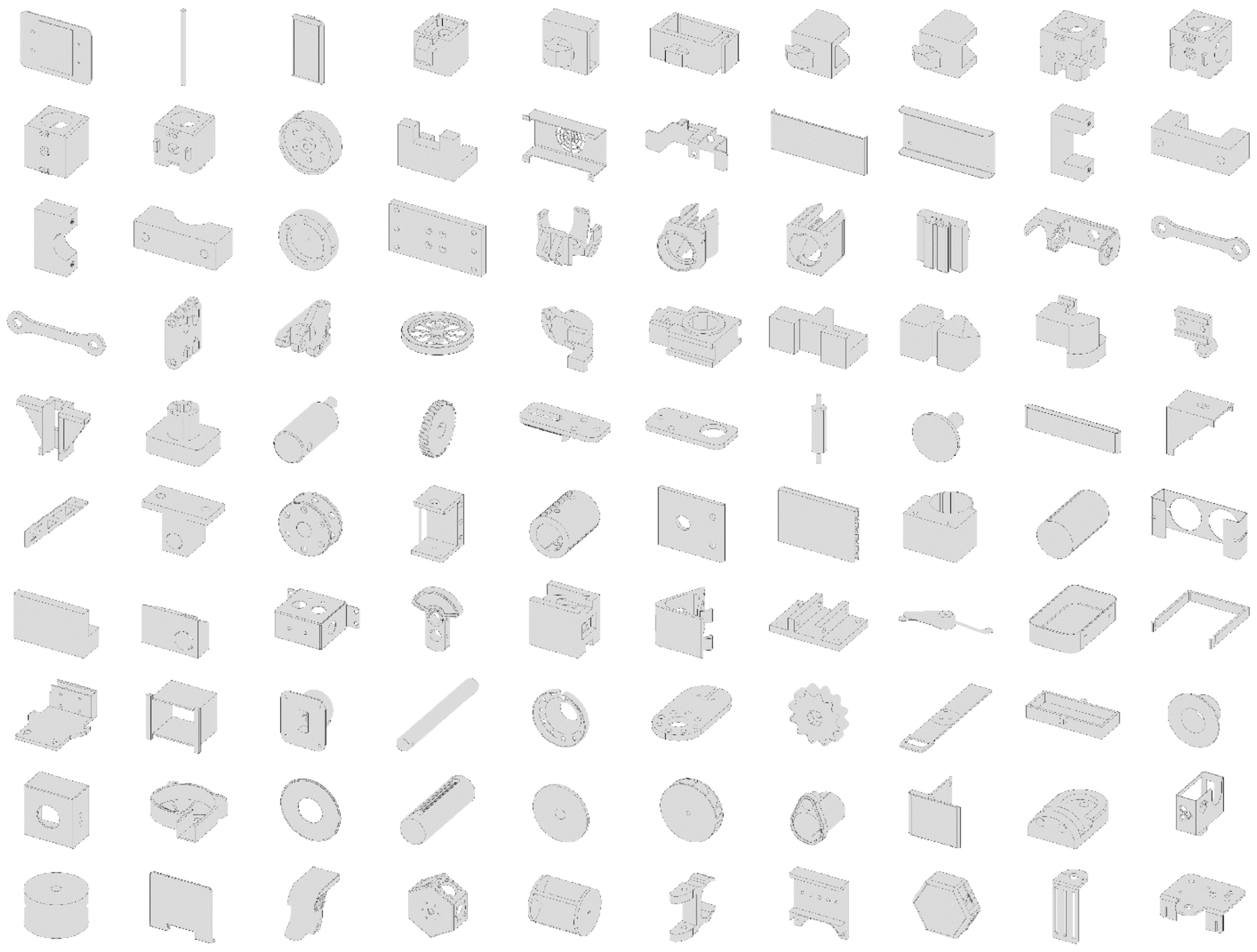


Figure 36 Examples of the Phenix Automation dataset.

3D data can be represented in many forms like point clouds, voxels, and mesh. But for the task of the robot gripper, representing the 3D object in the form of point clouds will not help the task in hand. Since point clouds are sparse, it just provides the information about various 3D points and little about the relationship between those points. To use point clouds for this task, the relationship between different points should be discovered and then points suitable for robot gripping should be identified. There exists ambiguity in the case of edges. For example, should a point on the edge be considered for robot gripping? Similar problems arise if 3D data is represented in the form of voxels, since each voxel just stores the information of 3D points belonging to that voxel, and as such, this information may not be useful for this task. Representing 3D data in the form of a 3D surface mesh provides more information of the 3D object like faces and neighboring connectivity. Since faces provide rich surface information of the 3D object, using the information of faces can help identify the good faces for our robot gripper's suction cup to adhere to.

Examples of different 3D CAD models fetched online are shown in Figure 36. CAD models fetched meet the real-world distribution of different industrial machinery parts. We generated 3D mesh from the CAD model using the MATLAB PDE toolbox, and annotated good and bad faces. We created a MATLAB tool for faster dataset annotation.

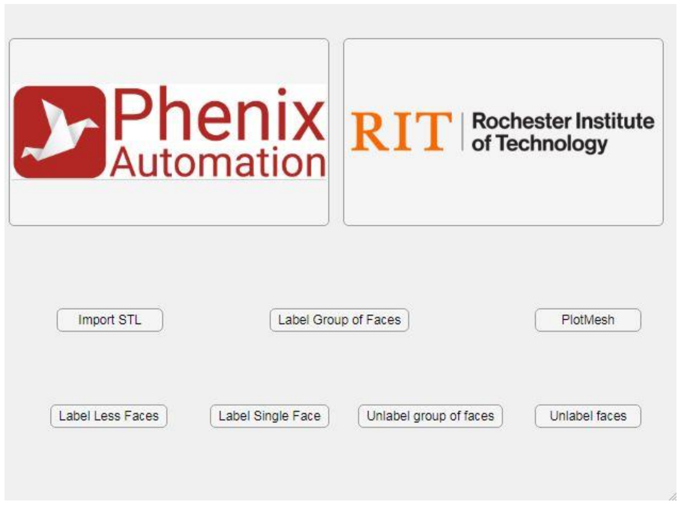


Figure 37 View of MATLAB tool for dataset annotation.

The MATLAB tool developed for dataset annotation purpose is shown in Figure 35. All the 3D CAD models fetched online are converted into the MATLAB supported format STL (Stereolithography). We used the PDE (Partial Differential Equations) toolbox in MATLAB to generate tetrahedral 3D mesh surfaces, since we are interested only on outside body surfaces. We converted tetrahedral 3D mesh into triangular 3D meshes and annotated good vs. bad faces using MATLAB callback functions.

In the MATLAB tool shown in Figure 37, after we import the STL file, the tool will automatically generate a triangular 3D mesh of the 3D object, and then user can annotate all the good faces of the 3D object. A user can label group of faces or single depending on the requirement.

Examples of training data are shown in Figure 38. The color red corresponds to bad faces and the color cyan corresponds to good faces. A good face should be a minimum area of 10 mm both on *X* and *Y* axis. Some examples in Figure 38 don't contain any good faces for the robot gripper to hold. All the training data has different shapes. Some parts contain more triangular elements (faces) and some parts contain lesser triangular elements.

Training Data

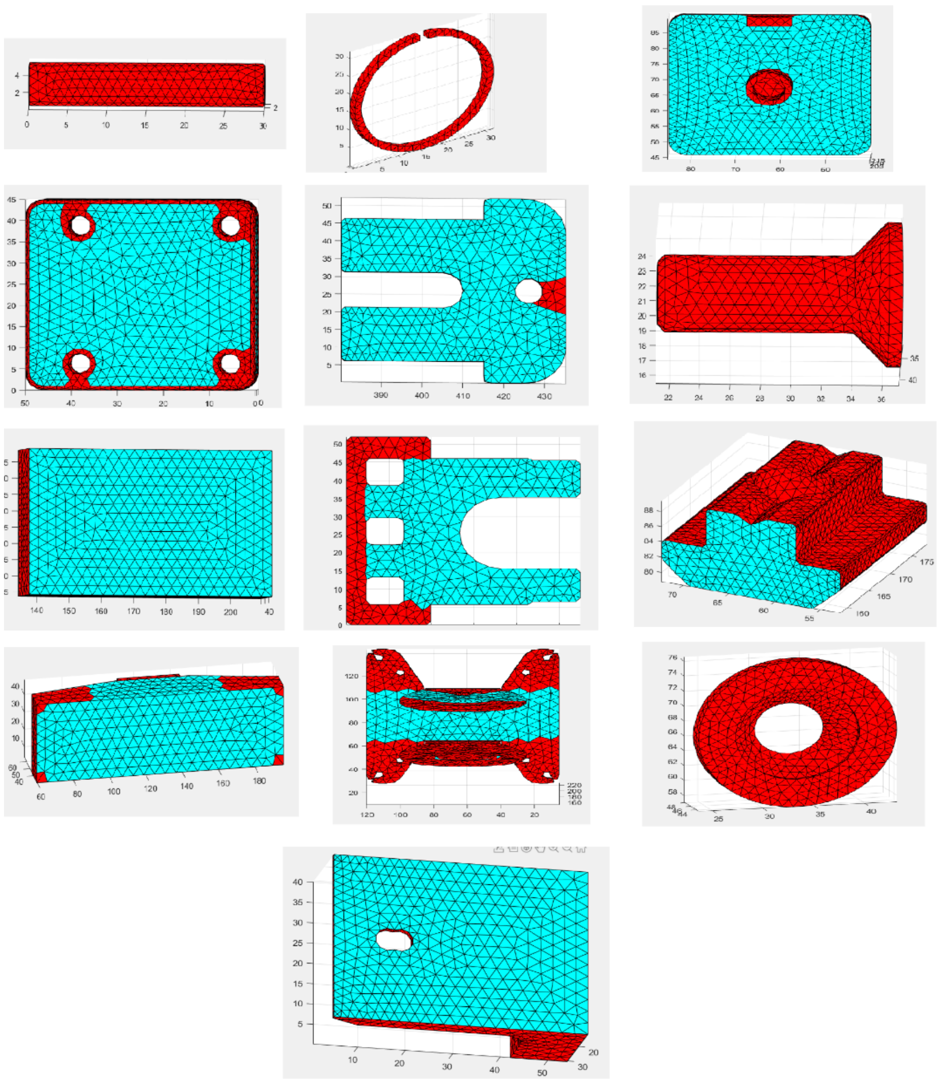


Figure 38 Examples of annotated dataset for robot gripper.

Chapter 5

Results and Analysis

5.1 Results

A. MS PCN:

Table 1 Results of MS PCN with no voxels.

Model	CD (Complete PCD)	EMD (Coarse PCD)
PCN (Baseline)	0.00986	0.05631631
MS PCN (no voxels)	0.0150	0.07454845

In Table 1, the results are shown for the baseline results and multistage decoder with encoder same as the baseline. The metric compared is the average Chamfer Distance (CD) loss of the test dataset. The encoder for the model in Table 1 didn’t contain any voxel information, it is the same as the encoder used in the baseline model. In the decoder used in the model, the complete point cloud is optimized in the coarse stage, the middle stage, and the complete stage. The complete point clouds and the coarse point clouds loss is compared with the baseline results. For the complete point clouds, the CD is used to compare with the baseline and for the coarse point clouds, and the Earth Mover Distance (EMD) loss function is used. Optimizing the complete point cloud in multiple stages didn’t help the model to reach results better than baseline results. We trained our models with different hyperparameters like changing the learning rate schedule, initial learning rate, different optimizers, and different batch sizes. We provided our best results obtained during training. The inference time for the model without voxels is 0.11 milli seconds.

Table 2 Results of MS PCN.

Model	CD (Complete PCD)	EMD (Coarse PCD)
PCN (Baseline)	0.00986	0.05631
MS PCN	0.013018	0.06262

In Table 2, the results are compared between baseline and the model with voxelized encoder and multistage decoder. For the encoder in the model, the input point cloud is voxelized and the voxel features are extracted to concatenate with the global features. And the decoder is the same as the model used for comparison in Table 1. The complete point cloud output from the decoder is optimized in multiple stages. Results show an improvement compared with the model in Table 1.

Adding voxel features helps the model to improve the coarse and the complete point clouds. The voxel features are extracted by using Voxel Feature Extraction layers (VFE) which contains 2D convolutions. These 2D convolutions help the points in different voxels to interact and the resultant global feature vector of the encoder is a much better global understanding of the model. We trained our models with different hyperparameters like changing the learning rate schedule, initial learning rate, different optimizers, and different batch sizes, we provided our best results obtained during training. The inference time for the model with voxels is 0.12 milli seconds.

B. **Edge convolution based point completion network**:

Table 3 Results of Edge convolution based PCN.

Model	CD (Complete PCD)	EMD (Coarse PCD)
PCN (Baseline)	0.00986	0.05631
Edge convolution PCN	0.14152	1.82143

Table 3 shows the results for edge convolution based point completion network. The encoder for the model consists of edge based convolutions to extract the global features concatenated with the voxel-based global feature vector. We have used edge based convolution to extract local features more robustly. To extract the voxel features we have used 7 x 7 convolutions, 5 x 5 based convolutions, 3 x 3 based convolutions, 1 x 1 based convolutions, and fused all these features to get the final vector. The complete point cloud output from the decoder is optimized in multiple stages. In the middle stage instead of using replicated coarse features, we have used interpolated features, similarly for the final output stage. We approached in this way since interpolated features propagate in a much better than replicated features from the previous layers. Unfortunately, the results are not as good compared to our previous model and baseline model. The edge based convolution is good at extracting the local features in the complete point cloud since the input is a partially complete point cloud, but the model is not able to extract features much affectively. Adding the voxel features by extracting in the way we mentioned before didn't help the model for optimizing the coarse point cloud as well. We trained our models with different hyperparameters like changing the learning rate schedule, initial learning rate, different optimizers, and different batch sizes, we provided our best results obtained during training. We conducted many experiments by adding the only single component to the baseline model and tested our results.

Table 4 Results of Edge convolution with voxels no multistage.

Model	CD (Complete PCD)	EMD (Coarse PCD)
PCN (Baseline)	0.00986	0.05631
Edge convolution with voxels no multistage	0.155976	2.20802

Table 4 shows the results of the model with the encoder same as the model in Table 3 but with a modified decoder. To extract the global features from the encoder, we used edge based convolutions concatenated with the voxel feature extraction layers. For extracting the voxel features we used the same method as mentioned before. The decoder in the model doesn't contain multiple stages, it has a coarse stage and then the final stage, same as in the baseline model. Instead of concatenating the replicated features of the coarse point cloud, we used interpolated features of the coarse point cloud. Removing the middle stage of the decoder made the model increase the loss both for the coarse point cloud and the complete point cloud.

To test whether edge convolutions are really helping to complete the partial point clouds, we tried optimizing only the coarse point cloud (no middle and no complete point clouds). The input is a partial point cloud, and the output is a coarse point cloud. The results are shown in Table 5.

Table 5 Results of Edge convolution single stage PCN.

Model	CD (Chamfer Distance)
PCN (Baseline)	0.00986
Edge convolution single stage	0.162826

Results for the edge convolution single stage model are shown in Table 5. We can clearly observe that edge convolutions are not helping the model to optimize the coarse point cloud. So, using edge convolutions in the point completion task doesn't add any advantage.

Table 6 Results of Baseline encoder + voxels and single stage interpolated decoder.

Model	CD (Complete PCD)	EMD (Coarse PCD)
PCN (Baseline)	0.00986	0.05631
Baseline encoder + voxels and single-stage interpolated decoder	0.15523	3.28933

Table 6 shows the results of the baseline model encoder with voxels and no multistage. The encoder of the model contains the baseline model encoder and voxel feature extraction layers. To extract the voxel features we have used an Inception-style architecture as mentioned before. The decoder consists of the coarse stage and the final stage. Instead of the replicated features of the coarse point cloud we have concatenated the interpolated features at the final stage. There is no improvement for the complete point cloud and the coarse point cloud is much worse compared to the models shown in Table 3 and Table 4. Extracting the voxel features in Inception-style architecture is not helping the model to optimize both the coarse and the complete point clouds.

Table 7 Results of Baseline encoder with single stage interpolated decoder.

Model	CD (Complete PCD)	EMD (Coarse PCD)
PCN (Baseline)	0.00986	0.05631
Baseline encoder + Single stage interpolated decoder	0.153162	0.19323

Table 7 shows the results of the model with the encoder the same as the baseline model with no voxels and no multistage. The encoder of the model is identical to the baseline model. The decoder consists of the coarse stage and the final stage. Instead of the replicated features of the coarse point cloud we have concatenated the interpolated features. By removing the voxel features from the encoder there is a bit of improvement for the coarse point cloud output when compared with Table 6 and Table 3. But there is no improvement for the complete point cloud output. Extracting the voxel features in Inception-style architecture does not appear to be adding information of the input partial point clouds to optimize the coarse and the complete point cloud.

Table 8 Results of Baseline encoder with a multistage interpolated decoder.

Model	CD (Complete PCD)	EMD (Coarse PCD)
PCN (Baseline)	0.00986	0.05631
Baseline encoder + Multistage interpolated decoder	0.14680	0.23593

Table 8 shows the results of the model with the encoder the same as the baseline model with no voxels and multistage interpolated decoder. The encoder of the model contains just as the baseline model as adding the voxel features in Inception-style architecture does not appear to be extracting additional features from the input partial point cloud. We didn't extract voxel features; we used the same encoder as the baseline model. The decoder is a multistage decoder consisting of the coarse stage, the middle stage, and the final stage. Instead of the replicated features at the middle stage and the final stage, we used interpolated versions. At the middle stage, we used interpolated features of the coarse point cloud and at the final stage, we used interpolated features of the middle point cloud. Adding a multistage decoder helped the model to improve the results of the complete point cloud compared to the models used in Table 4, Table 6, and Table 7. But there is no improvement for the coarse point cloud output. But the coarse point cloud results are much better compared to the models with voxel features extracted in Inception-style architecture.

C. Capsule based decoder:

Table 9 Results of Capsule based decoder.

Model	CD (Chamfer Distance)
PCN (Baseline)	0.00986
Capsule based decoder	0.02061

Table 9 shows the results of the point completion network with the capsule-based decoder. The loss of the model with the capsule-based decoder is high compared to the baseline results. In the capsule-based model, we are optimizing the coarse point cloud by applying a multilayer perceptron on the codeword from the encoder and the complete point cloud by employing the dynamic routing algorithm. Because of this multitasking setup, the weights learned using the dynamic routing algorithm are potentially getting disturbed. And the final complete point cloud is not generated by an end-to-end dynamic routing algorithm from the codeword, initially, an intermediate point cloud of shape *2048 x 3* is obtained by dynamic routing algorithm and the complete point cloud is obtained by folding operation from the intermediate coarse point cloud.

Table 10 Results of Capsule based architecture for coarse point cloud optimization.

Model	EMD (Coarse PCD)
PCN (Baseline)	0.05631
Capsule based architecture	0.06537

Table 10 shows the results for the model which uses capsule-based architecture to optimize the coarse point clouds. We changed the intermediate point cloud shape in the model used in Table 9 to the coarse point cloud shape. The coarse point cloud is comparable to the model shown in Table 1 but the results are not better compared to the baseline model. If the architecture has an end-end dynamic routing algorithm for optimizing the final point cloud, results would perhaps be better. The inference time for the model with capsule-based architecture is 0.65 milliseconds. Due to the dynamic routing algorithm, the inference time for this model is high compared to the previous models.

D. Multiview PCN:

Table 10 Results of Multiview PCN.

Model	CD (Chamfer Distance)
PCN (Baseline)	0.00986
DGCNN Multiview	0.0176

The results of the multiview PCN are shown in Table 10. The input to the encoder of the multiview PCN is the input partial clouds rotated by 0 degrees, 1 degree and 3 degrees, and use edge convolution to extract the features, then concatenated all the features with the voxel features. The decoder is the same as the MS PCN model, but our results unable to outperform the baseline results. Rotating the point cloud by different angles didn’t help the model to learn new features to complete the point cloud. The model can learn the same features in the rotated point clouds even with edge convolutions.

E. Self-supervised model:

Table 11 Results of Self-supervised PCN.

Model	EMD (Coarse PCD)
PCN (Baseline)	0.054201
Self-supervised model	0.054829

Table 11 results of self-supervision application to the point completion task. The pretext task used in this model is to categorize the input partial point cloud. The model learned on these features is a good starting point for weights for the point completion task. The dataset used for this model is 40% less compared to other models. We used those weights to optimize the coarse point clouds only and compared them with baseline in Table 11. The results clearly show even though with less data the model can perform almost similarly. The model weights trained on the pretext task helped the model for the problem of point completion network.

We can use the models trained on this dataset to perform inference on KITTI dataset [61]. The KITTI dataset [61] provides the point clouds for a LiDAR data, we can use these several models to complete a point cloud of any object in the KITTI [61] lidar point clouds.

F. PointNet-Center:

For the 3D mesh segmentation task, we have trained different architectures. The results for all these architectures are shown in the sections *F*, *G*, *H* and *I*.

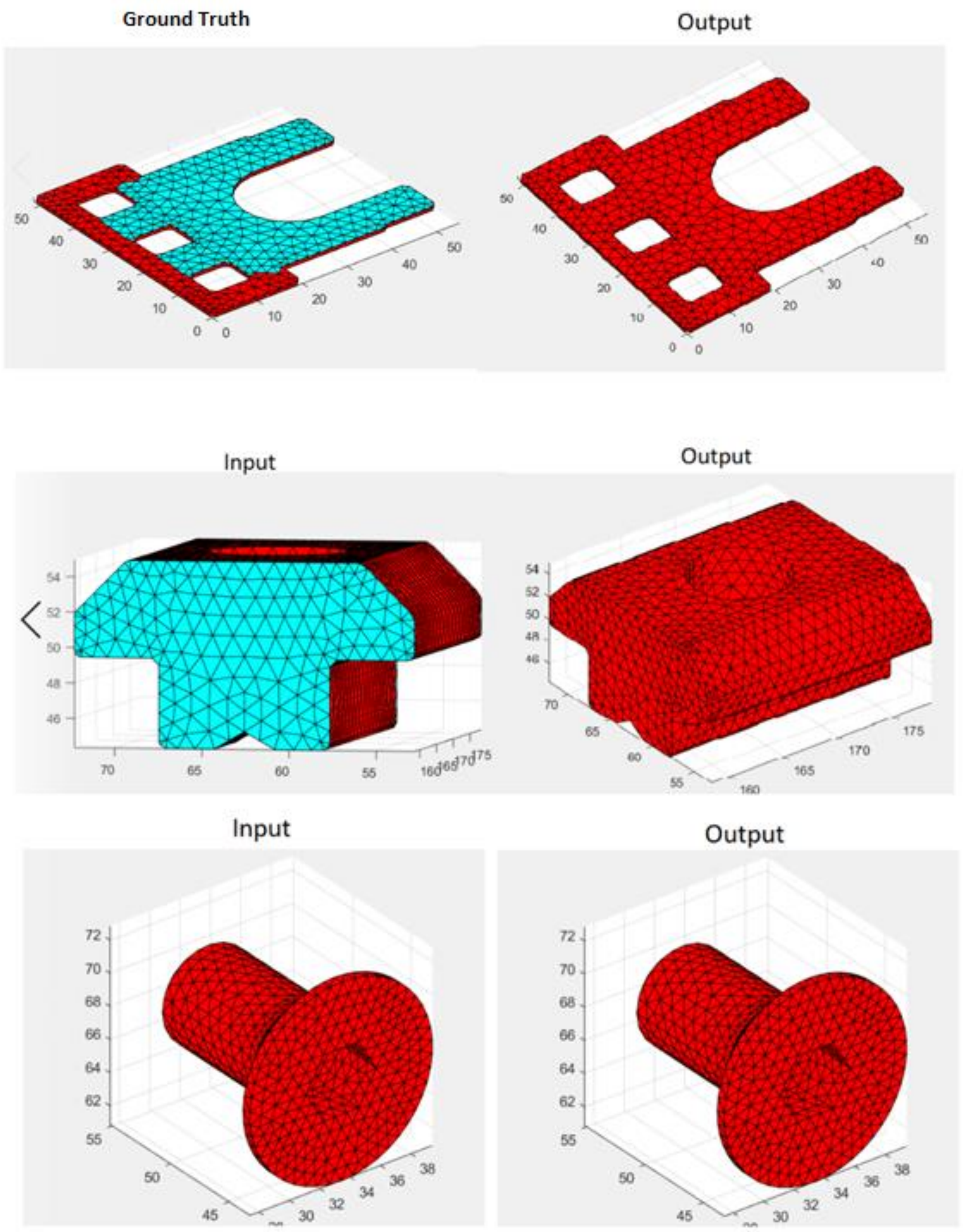


Figure 39 Results of PointNet-Center.

The results of PointNet-Center for 3D mesh segmentation is shown in the Figure 39. The dataset used to train the model is shown in Figure 38. The input to the model is the centers of each mesh in the input dataset. The model is not able to distinguish between good face and bad face, it always outputs every face of the mesh as bad for gripping. In the Figure 40, the ground truth and the corresponding outputs are shown. Many for the input samples having good faces, the model outputs every face as bad face. Since the PointNet [15] architecture just learns the pointwise features, this information alone is not sufficient to segment the input 3D mesh.

G. PointNet-Mesh:

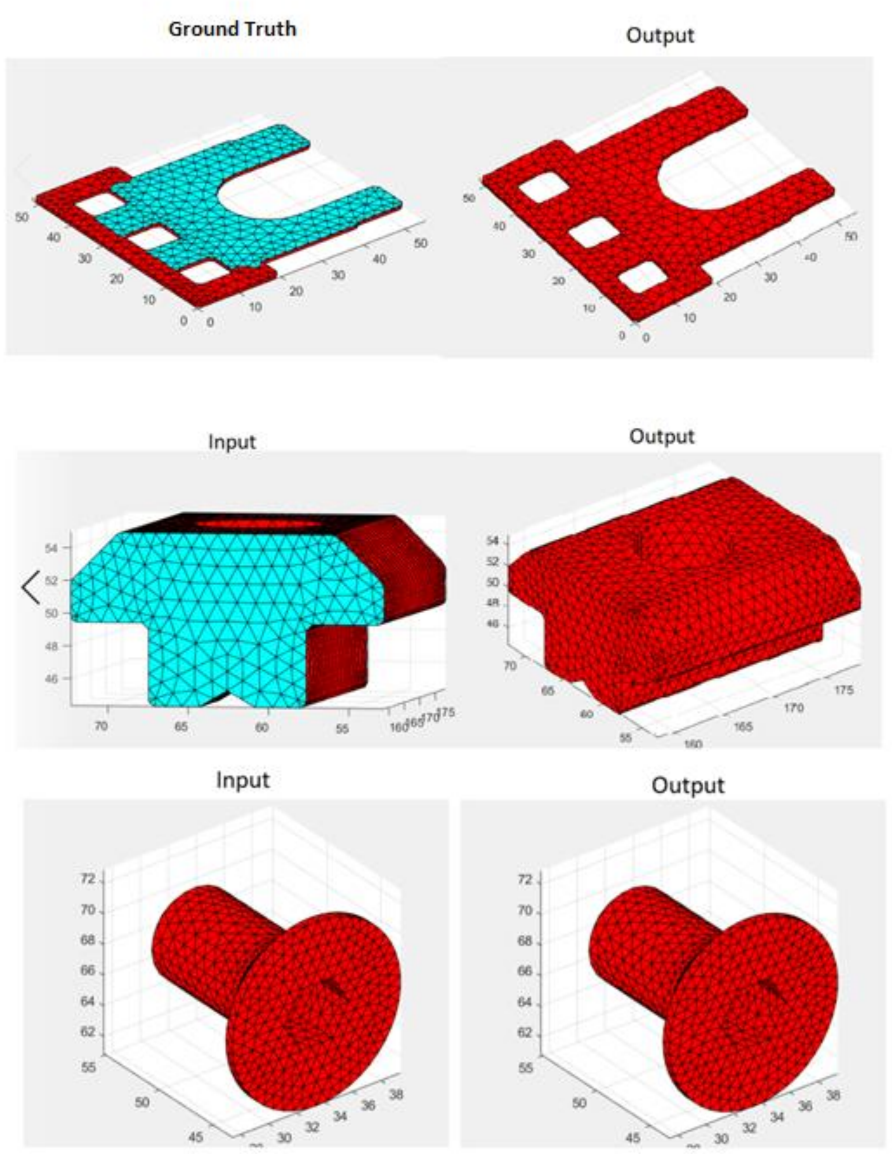


Figure 40 Results of PointNet-Mesh.

The results of 3D mesh segmentation for the model PointNet-Mesh is shown in Figure 40. The input to the model is a concatenated vector of all corners in each face of the 3D input mesh. Even though the input has many samples having good faces, the model always outputs each face as a bad face. Even though the input has information about each mesh, but the PointNet [15] style architecture unable to learn the relation between the corners of each face and different faces in the input 3D mesh.

H. DGCNN-Center:

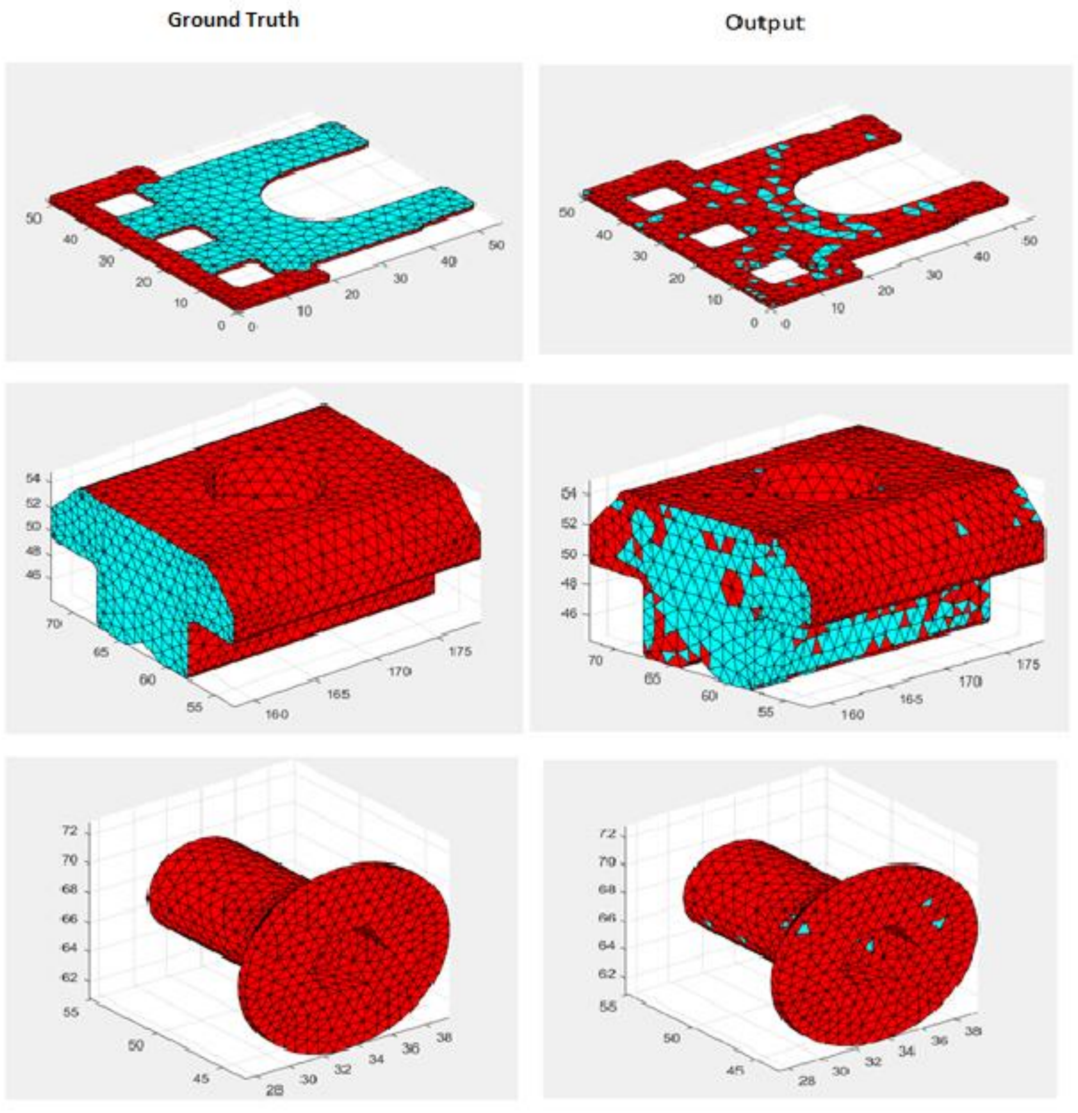


Figure 42 Results of DGCNN-Center.

The results of the model DGCNN-Center are shown in Figure 42. The input to the model is the centers of each mesh obtained from the 3D mesh shown in Figure 39. Unlike PointNet [15] style models, DGCNN [16] based models are able to output some good faces. Even though the dataset is small, the model is able to learn the relation between different faces in the 3D mesh to classify good and bad faces. Since [16] constructs a *k*-nearest graph, it extracts the local neighborhood information much better than PointNet [15] style based architectures.

I. MeshNet:

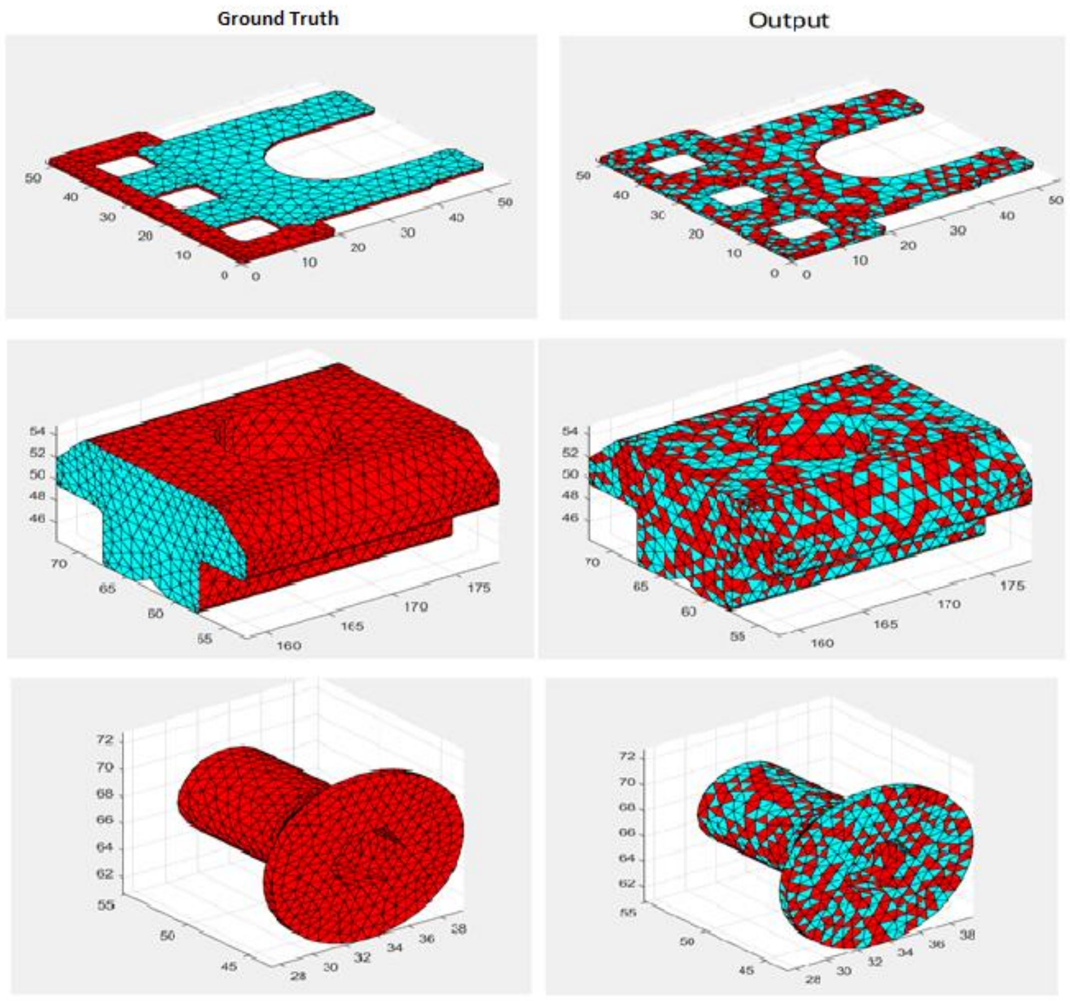


Figure 43 Results of MeshNet.

The results of the MeshNet is shown in Figure 43. We used the same architecture as in [48]. The input to the model contains face information, normal of each face and neighbors of each face. With the limited training dataset, the model is more inclined to output most of the faces as a good face. In Figure 43, we can see even for the input sample which doesn't contain any good faces, the model predicted many good faces. DGCNN-Center has better results compared to the MeshNet, even though MeshNet considers each face as the operating unit and applies mesh convolutions. The DGCNN-Center model is able to learn the relation between different faces with just information about the centers.

Chapter 6

Conclusions

6.1 Conclusions

This thesis presents the different architectures for the task of point completion and provides the basis for identifying good and bad faces of a robot gripper. This work provides different effective ways of extracting the features of a partial point cloud and provides an analysis of the different components of the architectures. Adding the voxel information of the point cloud in the right way definitely helps in better feature extraction. Even though edge convolutions are very effective in local feature extraction, edge convolutions alone are not sufficient for the task of point completion. This work shows the application of the self-supervision concept for the task of point completion. Training the model on a pretext task provides a good initialization for the model’s weights and can achieve similar results with fewer data. For the robot gripper task, formulating the problem as 3D mesh segmentation will provide the good and bad faces of a 3D object.

6.2 Future Work

This work explores PointNet based architectures for the tasks of point completion and 3D mesh segmentation. Some of the possible extensions of this work are:

- For extracting the partial point cloud features, try graph based approaches instead of PointNet based architectures.
- For extracting the voxel level features, a 3D convolution can be used instead of 2D convolutions.
- Extending the idea of dynamic routing to an end-end capsule-based architecture.
- Extending the idea of self-supervision for the 3D mesh segmentation problem.

- Dynamic Graph CNN showed better results for 3D mesh segmentation. As such, exploring different architectures of graph-based networks for the problem of 3D mesh segmentation would be helpful.
- Evaluate the performance of different point cloud completion networks with the other datasets.

Bibliography

[1] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On visual similarity based 3d model retrieval. In Computer graphics forum, volume 22, pages 223–232. Wiley Online Library, 2003.

[2] Yuan, Wentao, et al. "Pcn: Point completion network." 2018 International Conference on 3D Vision (3DV). IEEE, 2018.

[3] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition.

[4] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pages 3212–3217. IEEE, 2009.

[5] Y. Li, R. Bu, M. Sun, and B. Chen. Pointcnn. arXiv preprint arXiv:1801.07791, 2018.

[6] M. Aubry, U. Schlickewei, and D. Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on, pages 1626–1633. IEEE, 2011.

[7] X. Han, Z. Li, H. Huang, E. Kalogerakis, and Y. Yu. Highresolution shape completion using deep neural networks for global structure and local geometry inference. arXiv preprint arXiv:1709.07599, 2017.

[8] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1912–1920, 2015.

[9] D. Stutz and A. Geiger. Learning 3d shape completion from laser scan data with weak supervision.

[10] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 2016.

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203, 2013.

[12] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In Proceedings of the IEEE International Conference on Computer Vision Workshops, pages 37–45, 2015.

[13] H. Ling and D. W. Jacobs. Shape classification using the inner-distance. IEEE transactions on pattern analysis and machine intelligence, 29(2):286–299, 2007.

[14] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In Proc. NIPS, 2017.

[15] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In Proc. CVPR, 2017.

[16] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. arXiv preprint arXiv:1801.07829, 2018.

[17] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3384–3391. IEEE, 2008.

[18] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H.Yang, and J. Kautz. Splatnet: Sparse lattice networks for point cloud processing. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages
2530–2539, 2018.

[19] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In IEEE/RSJ International Conference on Intelligent Robots and Systems, September 2015.

[20] J. Varley, C. DeChant, A. Richardson, J. Ruales, and P. Allen. Shape completion enabled robotic grasping. In Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on, pages 2442–2447. IEEE, 2017.

[21] B. Yang, S. Rosa, A. Markham, N. Trigoni, and H. Wen. 3d object dense reconstruction from a single depth view. arXiv preprint arXiv:1802.00411, 2018.

[22] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds.

[23] H. Fan, H. Su, and L. Guibas. A point set generation network for 3d object reconstruction from a single image. In Conference on Computer Vision and Pattern Recognition (CVPR), volume 38, 2017.

[24] J. Sun, M. Ovsjanikov, and L. Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In Computer graphics forum, volume 28, pages 1383–1392. Wiley Online Library, 2009.

[25] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. IEEE Transactions on pattern analysis and machine intelligence, 21(5):433– 449, 1999.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in Advances in Neural Information Processing Systems 25, F.

Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[27] C. Szegedy et al., “Going Deeper With Convolutions,” presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.

[28] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[29] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” arXiv:1207.0580 [cs], Jul. 2012.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09, 2009.

[31] Karen Simonyan and Andrew Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.

[32] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580–587.

[33] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

[34] Yang, Yaoqing, et al. "Foldingnet: Point cloud auto-encoder via deep grid deformation." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

[35] Gidaris, Spyros, Praveer Singh, and Nikos Komodakis. "Unsupervised representation learning by predicting image rotations." arXiv preprint arXiv:1803.07728 (2018).

[36] Zhou, Yin, and Oncel Tuzel. "Voxelnet: End-to-end learning for point cloud based 3d object detection." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

[37] Yifan, Wang, et al. "Patch-based Progressive 3D Point Set Upsampling." arXiv preprint arXiv:1811.11286 (2018).

[38] Mandikal, P., and Babu, R. V. 2019. Dense 3d point cloud reconstruction using a deep pyramid network. In Winter Conference on Applications of Computer Vision (WACV).

[39] https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/1d-convolution-block.

[40] https://github.com/PetarV-/TikZ/tree/master/2D%20Convolution

[41] https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

[42]https://software.intel.com/sites/products/documentation/doclib/daal/daal-user-and-reference-guides/daal_prog_guide/GUID-9B434D4F-C723-4191-9A88-69148C75A3F1.htm

[43] https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

[44] https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html

[45] https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/

[46] Zhao, Yongheng, et al. "3D point capsule networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

[47] Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." Advances in neural information processing systems. 2017.

[48] Feng, Yutong, et al. "MeshNet: mesh neural network for 3D shape representation." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. 2019.

[49] Dai, Angela, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. "Scannet: Richly-annotated 3d reconstructions of indoor scenes." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5828-5839. 2017.

[50] Armeni, Iro, et al. "Joint 2d-3d-semantic data for indoor scene understanding." arXiv preprint arXiv:1702.01105 (2017).

[51] https://grabcad.com/

[52] LaLonde, Rodney, and Ulas Bagci. "Capsules for object segmentation." arXiv preprint arXiv:1804.04241 (2018).

[53] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

[54] Groueix, Thibault, et al. "A papier-mâché approach to learning 3d surface generation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[55] Lang, Alex H., et al. "Pointpillars: Fast encoders for object detection from point clouds." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

[56] Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks." In CVPR, vol. 1, no. 2, p. 3. 2017.

[57] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[58] Zhang, Richard, Phillip Isola, and Alexei A. Efros. "Colorful image colorization." European conference on computer vision. Springer, Cham, 2016.

[59] Noroozi, Mehdi, and Paolo Favaro. "Unsupervised learning of visual representations by solving jigsaw puzzles." European Conference on Computer Vision. Springer, Cham, 2016.

[60] Pathak, Deepak, et al. "Context encoders: Feature learning by inpainting." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
[61] Geiger, Andreas, et al. "Vision meets robotics: The kitti dataset." The International Journal of Robotics Research 32.11 (2013): 1231-1237.