

Otto-von-Guericke-University Magdeburg

# Project

## Mesh Segmentation of Aneurysms using Geometric Deep Learning

by: Jayashri Masilamani

## **Abstract**

Deep learning is the sub-field of AI algorithms that have shown the state of the art performance in fields like computer vision. Geometric deep learning is the set of deep learning algorithms that work on non-euclidean data like graphs and meshes. One of the recent literature called “Mesh CNN: A Network with an Edge” is specially designed for the 3D triangular meshes that have shown impressive results on the part segmentation task. The central idea of the thesis is to perform the segmentation task using Mesh CNN on the available intracranial aneurysm dataset. Firstly, a pipeline is created for the edge-based ground-truth generation for the intracranial aneurysm dataset. One of the limitations of the adaptation of deep learning algorithms for medical data is the lack of enough data for the learning of the network. A simple data augmentation pipeline was developed to increase the number of aneurysm training samples. Then, various experiments are conducted for different hyper-parameters and the results of the experiments are gathered. Finally, an exhaustive analysis is done on the experimental results based on the performance of the algorithm for different hyperparameter values of the network.

**Task of the Thesis in the Original:**

## **Declaration by the candidate**

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been marked.

The work has not been presented in the same or a similar form to any other testing authority and has not been made public.

Magdeburg, August 11, 2020

# Contents

<b>List of Figures</b>	<b>5</b>
<b>List of Tables</b>	<b>7</b>
<b>1. Introduction</b>	<b>8</b>
1.1. Meshes and efficiency . . . . .	8
1.2. 3D Deep learning . . . . .	9
1.3. Challenges . . . . .	11
1.4. Project description . . . . .	11
1.5. Structure of the thesis . . . . .	11
<b>2. Background</b>	<b>13</b>
2.1. Deep learning . . . . .	13
2.1.1. Artificial Neural Network (ANN) . . . . .	13
2.1.2. Convolutional Neural Network (CNN) . . . . .	13
2.1.3. Reasons for the rise of Deep learning . . . . .	15
<b>3. Literature Review</b>	<b>17</b>
3.1. Classical Mesh Segmentation algorithms . . . . .	17
3.1.1. Segmentation Types . . . . .	17
3.1.2. Partition Criterion . . . . .	17
3.1.3. Segmentation Techniques . . . . .	18
3.2. Segmentation of 3D Deep Learning . . . . .	19
3.2.1. Introduction . . . . .	19
3.2.2. 3D Data . . . . .	20
3.2.3. Euclidean data . . . . .	21
3.2.4. Non-Euclidean data . . . . .	22
<b>4. Methodology</b>	<b>28</b>
4.1. Introduction . . . . .	28
4.1.1. Input Features . . . . .	28
4.2. MeshCNN Network . . . . .	29
4.2.1. Mesh Convolution . . . . .	29
4.2.2. Mesh Pooling . . . . .	30
4.2.3. Mesh Un-pooling . . . . .	32

4.3. Segmentation Network in MeshCNN . . . . .	32
4.3.1. Mesh U-Net . . . . .	32
4.4. Augmentation . . . . .	33
4.5. Segmentation Experiments . . . . .	34
<b>5. Implementation</b>	<b>37</b>
5.1. Introduction . . . . .	37
5.2. Concept of segmentation of aneurysm . . . . .	37
5.3. Preparation of Dataset . . . . .	38
5.3.1. Ground-truth files . . . . .	39
5.3.2. Data Pre-processing . . . . .	40
5.3.3. Vertice grouping . . . . .	40
5.3.4. Workflow of ground-truth generation . . . . .	43
5.4. Augmentation of Aneurysm . . . . .	44
5.4.1. Introduction . . . . .	44
5.4.2. Steps of Augmentation method . . . . .	45
5.4.3. Cleaning of the new augmented file in MeshLab . . . . .	46
5.4.4. Conclusion . . . . .	46
<b>6. Experiments and results</b>	<b>48</b>
6.1. Metrics . . . . .	48
6.2. Hyperparameters list . . . . .	48
6.3. Human Segmentation Experiment . . . . .	49
6.4. Aneurysm Experiment . . . . .	50
6.4.1. Experiment 1 . . . . .	50
6.4.2. Experiment 2 . . . . .	52
6.4.3. Experiment 3 . . . . .	53
6.4.4. Experiment 4 . . . . .	53
6.4.5. Experiment 5 . . . . .	56
<b>7. Discussion</b>	<b>58</b>
7.1. Challenges and solutions . . . . .	58
7.1.1. Ground-truth data generation . . . . .	58
7.1.2. Hyperparameter tuning and training time . . . . .	58
7.2. Interpretation of results . . . . .	59
7.2.1. Human segmentation experiment . . . . .	59
7.2.2. Aneurysm segmentation experiment . . . . .	59
<b>8. Conclusion</b>	<b>65</b>
8.1. Summary . . . . .	65

8.2. Future and Recommendations . . . . .	66
<b>Bibliography</b>	<b>68</b>
<b>A. Appendix: Training data of best experiment</b>	<b>71</b>
A.1. Experiment 3: Training loss . . . . .	71
A.2. Experiment 3: Test Accuracy . . . . .	72
A.3. Experiment 3: Parameters . . . . .	75

## **List of Acronyms**

**DSA** Digital Subtraction Angiography

**CTA** Computed Tomographic Angiography

**ANN** Artificial Neural Network

**CNN** Convolutional Neural Network

**Eseg** Edge Segmentation

**Seseg** Soft Edge Segmentation

**ReLU** Rectified Linear Unit

**FC** Fully Connected Layer

**CPU** Central Processing Unit

**GPU** Graphics Processing Unit

**MLP** Multi Layer Perceptron

**STN** Spatial Transformer Network

**RNN** Recurrent Neural Network

**GCN** Graph Convolutional Network

## List of Figures

1.1.	The structure of the aneurysm mesh . . . . .	8
1.2.	A sample of triangular mesh . . . . .	9
2.1.	Basic structure of an artificial neural network. . . . .	14
2.2.	The building blocks of convolutional neural network . . . . .	15
3.1.	Various types of the 3D-Data representation. . . . .	20
3.2.	Flow chart of volumetric segmentation. . . . .	22
3.3.	Flow chart of multi-view segmentation. . . . .	23
3.4.	Various types of the 3D-Data representation [6] . . . . .	25
3.5.	Various types of the 3D-Data representation. . . . .	25
4.1.	1-ring neighborhood of edge e. . . . .	29
4.2.	Input features of MeshCNN network. . . . .	30
4.3.	Mesh Pooling and Unpooling Operation. . . . .	31
4.4.	The building blocks of the Mesh U-net. . . . .	33
4.5.	Segmentation output of MeshCNN for the coseg dataset [1] . . . . .	34
4.6.	Segmentation output of MeshCNN for the Human segmentation dataset [1]	35
5.1.	Figure shows labels for aneurysm with at most five segments. . . . .	37
5.2.	Edge based segmentation file sample(eseg file). . . . .	40
5.3.	Face based segmentation file sample (Seseg file). . . . .	41
5.4.	Vertice grouping of aneurysm mesh sample in Blender software . . . . .	42
5.5.	Vertices labels are grouping for each of the 5 segments . . . . .	42
5.6.	Correctness of manual labelling is checked and the above figure confirm its correctness . . . . .	44
5.7.	Steps involved in manual augmentation . . . . .	45
5.8.	Change in size of clipped aneurysm sample . . . . .	46
5.9.	New augmented aneurysm samples. . . . .	47
6.1.	Test result of large Human segmentation test sample . . . . .	50
6.2.	Training loss curve (left) and test accuracy (right) of Experiment 1 . . . . .	51
6.3.	Test set Aneurysm samples of experiment 1 . . . . .	51
6.4.	Training loss curve (left) and test accuracy (right) of Experiment 2 . . . . .	52
6.5.	Test set Aneurysm samples of experiment 2 . . . . .	53

---

*LIST OF FIGURES*

---

6.6.	Training loss curve (left) and test accuracy (right) of Experiment 3 . . . . .	54
6.7.	Test set Aneurysm samples of experiment 3 . . . . .	54
6.8.	Training loss curve (left) and test accuracy (right) of Experiment 4 . . . . .	55
6.9.	Test set Aneurysm samples of experiment 4 . . . . .	55
6.10.	Training loss curve (left) and test accuracy (right) of Experiment 5 . . . . .	57
6.11.	Test set aneurysm samples of experiment 5 . . . . .	57
7.1.	Comparison of test samples (top row) and ground-truth (bottom row) of experiment 1 . . . . .	60
7.2.	Comparison of test samples (top row) and ground-truth (bottom row) of experiment 2 . . . . .	61
7.3.	Comparison of test samples (top row) and ground-truth (bottom row) of experiment 3) . . . . .	62
7.4.	Comparison of test samples (top row) and ground-truth (bottom row) of experiment 4 . . . . .	63
7.5.	Comparison of test samples (top row) and ground-truth (bottom row) of experiment 5 . . . . .	63

## List of Tables

4.1. COSEG Segmentation [3] . . . . .	35
4.2. Human Body Segmentation [3] . . . . .	36
6.1. Hyperparameters list . . . . .	49
6.2. Human segmentation . . . . .	50
6.3. Train/Test set . . . . .	50
6.4. Experiment 1 . . . . .	52
6.5. Experiment 2 . . . . .	52
6.6. Experiment 3 . . . . .	53
6.7. Experiment 4 . . . . .	56
6.8. Experiment 5 . . . . .	56

# 1. Introduction

Intracranial aneurysm is the condition of the localized dilation of the cerebral artery. The rupture and bleeding of the aneurysm could cause serious complications which may result in long-term neurological pathologies. The current gold standard for the diagnosis of the aneurysm is the digital subtraction angiography (DSA) which uses x-ray as the imaging modality. Recent times computed tomographic angiography (CTA) is the commonly used technique for aneurysm detection because of its advantages over DSA.

CTA is the minimally invasive procedure that sends the contrast agent through the patient intravenously. CT angiography consists of a CT scanner that captures the 3D images of the blood vessels in the region of interest. CTA is known for producing high-resolution images of blood vessels in the various parts of the body. Some of the diagnosis techniques also use Magnetic resonance (MR) imaging to capture the images of the blood vessel. The 3D triangular meshes of the aneurysm that are generated from the CTA modality are used in this thesis.

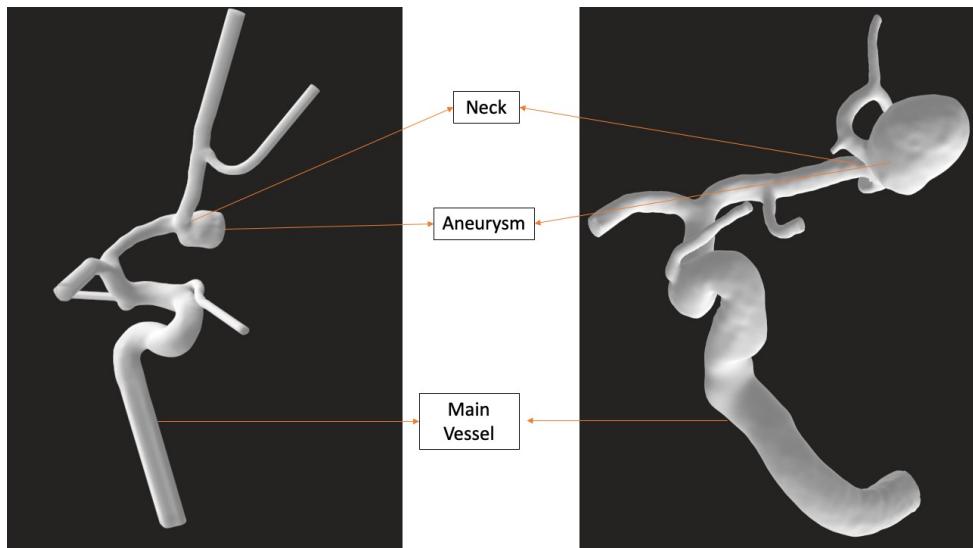


Figure 1.1.: The structure of the aneurysm mesh

## 1.1. Meshes and efficiency

The mesh structure is one of the common representation of 3D data. The mesh structure can be described as  $M = (V; F; E)$  with the set of vertices V, edges E and faces F. Mesh can be viewed as the graph also. The 3D-data described only with vertices are called

point clouds and the data defined only the faces of the mesh structure provides the surface information. Popularly used mesh structures are the polygonal meshes where the vertices and edges are joined such a way that produces the polygonal faces.

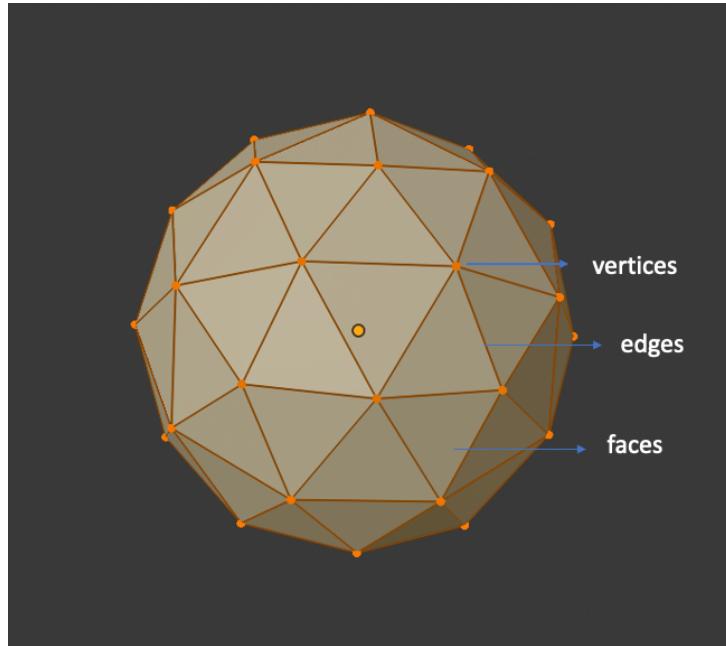


Figure 1.2.: A sample of triangular mesh

The triangular meshes are the meshes whose faces are triangle in shape as shown in figure 1.2 i.e there are exactly three edges that make up each face to form the triangular mesh. Meshes are considered as manifold when each edge is shared by at least two or more faces in the mesh structure. Manifold meshes have geometric properties that provide stable mathematical and structural information of the mesh. The aneurysm data used in this thesis are manifold and triangular meshes.

The major application of the 3D-mesh analysis is mesh segmentation. Mesh segmentation is the concept of partitioning the mesh structure into a number of sub-meshes where each sub-meshes has meaningful information in it. The segmentation can be based on geometric properties or semantic-based information of the mesh. The exhaustive literature related to the mesh segmentation is given in chapter 3.

## 1.2. 3D Deep learning

The central idea of this thesis is to take advantage of the recent advances in geometric deep learning algorithms and their capabilities. Deep learning is the sub-field of artificial intelligence whose architecture is loosely inspired by the neural network of the brain. The 2D-deep learning algorithms achieved state of the art performance on the various applications that use 1D/2D euclidean data like images, audio signals. But still, the deep

learning algorithms specially designed for 3D data are not as sophisticated as the 2D deep learning algorithms.

There are different types of 3D data types which can be made of the euclidean or non-euclidean structure. The transfer of 2D deep learning architectures and generalization techniques to 3D euclidean data-based architecture is straight-forward. The non-euclidean 3D-data like point cloud, graphs and meshes are widely available in recent times and it has a large number of applications in various fields like computer vision and medical image analysis. But the limitations to adapt the concepts from 2D deep learning operators like convolution to non-euclidean data types like meshes slows down the progress of 3D deep-learning algorithms.

The major reason for this limitation is because of the irregularity of the data structure, new convolutional techniques are required to make the deep learning operators work well with the non-euclidean data like point-clouds and meshes. Various publications were solely devoted to realizing the goal of developing such architecture that performs the classification and segmentation tasks on non-euclidean data. This architecture shows promising results that can be adapted and used on medical data that can potentially solve many medical data-related problems.

Our project deals with the 3D mesh aneurysm data. There are very few numbers of deep learning algorithms that take the mesh data as input and performs mesh classification and segmentation task exclusively. One of those few architectures is MeshCNN [1] which has a specialized convolutional neural network that was specifically designed for mesh edges as input data.

MeshCNN takes triangular meshes as input. The architecture is incorporated with the specialized mesh convolution and mesh pooling operators that take advantage of classical mesh analysis techniques like “edge collapse” to perform convolution and pooling tasks. The network takes the edges of the mesh as the building block of mesh and extracts a 5-dimensional feature vector to perform the task. The vertices of the mesh have no influence on the learning of the network. Experiments are conducted on MeshCNN to work on the benchmark mesh dataset and the results are evaluated for the classification and segmentation task.

The state of the art performance of MeshCNN on human segmentation [2] and the COSEG [3] dataset in the segmentation task motivated us to implement the MeshCNN on the medical data like aneurysm mesh. Adaptation of deep learning algorithms for medical applications is always a painstaking process because of the complexity of the medical data, limited access to the availability of medical data and the requirement of high standard results. It would be interesting to adapt the geometric deep learning algorithm like MeshCNN on the medical data and to see how well the algorithm can perform on the large-sized meshes like aneurysm i.e large number of input edges units.

### **1.3. Challenges**

The goal of the thesis is to generate the ground-truth data for the meshCNN and analyze the performance of the algorithm on the complex aneurysm data. The Aneurysm mesh sample is comparably larger than the mesh sizes of the benchmark dataset. The aneurysm mesh cannot be decomposed smaller than to some extent because of the need to preserve the structural information of the mesh.

The segmentation parts of each aneurysm are unique in contrast to the dataset used in the MeshCNN. It will be interesting to understand the robustness of the algorithm on a distinct dataset like Aneurysm.

The Mesh segmentation of aneurysm has many potential applications in research and development of aneurysm treatment.

### **1.4. Project description**

The fundamental part of the project is to generate a sufficient number of the ground-truth requirements by the MeshCNN to generate satisfactory segmentation results on the aneurysm data.

The exhaustive literature review has been conducted to understand the conventional mesh segmentation algorithm and the recent geometric deep learning algorithms for various 3D-data that shows promising results for segmentation tasks and documented.

A manual augmentation of two different mesh samples is performed and the new samples are labeled then, trained on the MeshCNN.

The limited number of the aneurysm data is increased with the adaptation of the augmentation technique with the motivation to improve the training performance and generalization of the network.

Various experiments are conducted for different hyperparameters and the segmentation outputs are visualized and results are analyzed. The hyperparameter values of various augmentation techniques are changed and the change in learning of the algorithm is analyzed.

Finally, some insights are made from the results of the aneurysm segmentation and few recommendations are suggested that would potentially improve the task performance.

### **1.5. Structure of the thesis**

The structure of this thesis is organized as follows:

Chapter 2 accounts for the fundamentals of deep learning followed by the exhaustive literature review of the classical mesh segmentation techniques and 3D deep learning concepts for segmentation tasks.

---

## *1. Introduction*

---

Chapter 3 is the methodology section that describes the building blocks of MeshCNN architecture and the segmentation experiments conducted on the paper

Chapter 4 gives the step-by-step information of the mesh ground-truth generation procedure

Chapter 5 is dedicated to the explanation of manual augmentation technique that was performed on the aneurysm task

Chapter 6 is about the list of experiments conducted on aneurysm dataset using MeshCNN and documentation of results.

Chapter 7 analyses the results of each experiment and the segmentation output that was generated.

Chapter 8 recommends the further improvements that can be made so that it may improve the performance of the aneurysm segmentation task.

Chapter 9 summarizes the various tasks that are completed during the thesis work.

## **2. Background**

### **2.1. Deep learning**

Deep learning is the subfield of the Artificial intelligence domain. The structure and working of deep learning are primarily inspired by the neural networks of the brain. The convolutional neural networks are the specialized form of artificial neural network which is designed for grid-like data such as images. CNN has achieved state of the art performance in image classification, segmentation, object recognition, and many other domains. In this section, we will see the basics of Artificial neural network (ANN) and then the basics and architecture of the convolutional neural network (CNN).

#### **2.1.1. Artificial Neural Network (ANN)**

It is also called Multi-layer perceptrons (MLPs). The “deep” in deep learning is because of the multiple hidden layers between the input and output layers. The network learns the features from the data that passes through the hierarchical layers. Each layer learns the features of an increasing level of complexity. ANN can learn complex functions because of the large number of hidden layers as it paves the way to do multiple levels of abstraction.

The early version of the neural networks are perceptrons which are single layer neural networks using linear activation functions. Perceptrons cannot model non-linear complex functions which are overcome by the introduction of more hidden layers and activation functions in the multi-layer perceptrons.

Figure 2.1 shows a simple ANN with one hidden layer and one output unit. Each node in the network consists of weights and activation functions. When the data passes from the input layer through hidden layers it performs multiplication of input with the weight value and non-linear function in each node thus progressively learns to approximate the underlying complex function in the data which can be evaluated in the output layer. This process of hierarchical learning is also called representational or feature learning.

#### **2.1.2. Convolutional Neural Network (CNN)**

CNN [4] is the improvised version of multi-layer perceptrons. It is specially designed for grid-like / two-dimensional inputs like images, audio signals, etc... CNN drops the previous structure of many hidden layers of neurons. CNN is inspired by the visual cortex of animals. The information in the images are highly related to the nearby pixels. Hence, CNN is designed to capture the connectivity information in the pixels of the image. CNN

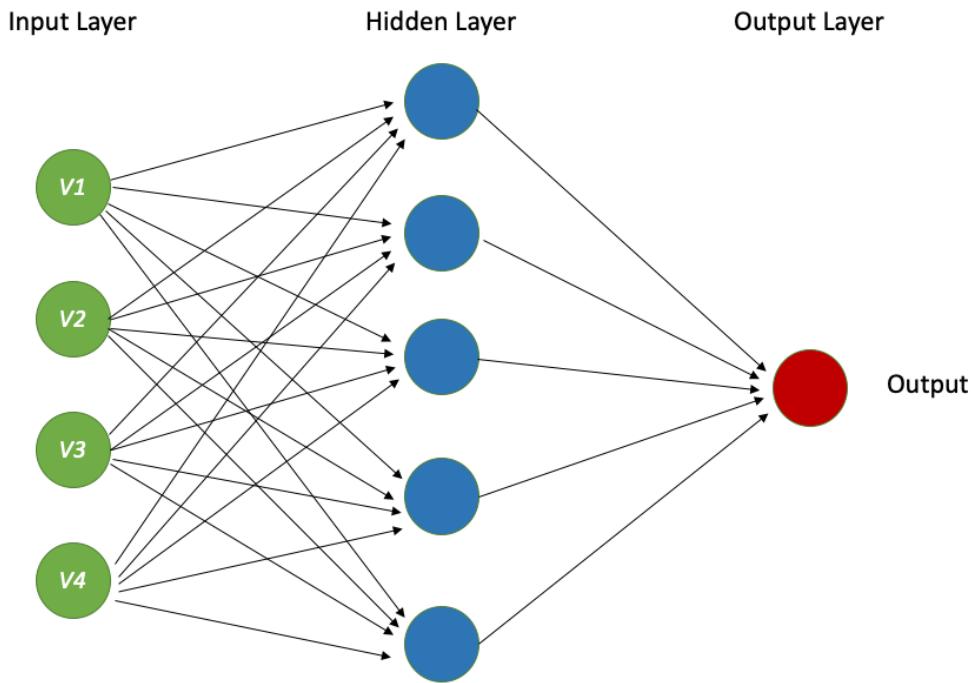


Figure 2.1.: Basic structure of an artificial neural network.

consists of special convolution and pooling layers that help the fast learning and reduces the number of parameters needed for training of the network.

The CNN consist of all or few of the following components:

**1. Convolutional Layer (CONV):** Convolutional layers can be viewed as kernels which will be applied to the group of pixels and performs a mathematical operation called convolution. Convolutional filters can be viewed as an activation maps.

**2. Rectified Linear Unit (ReLU):** ReLU is the non-linear activation function which is usually followed after the convolutional layer. ReLU is widely used and it is a better activation function compared to the previously used sigmoid and tanh functions.

ReLU reduces the vanishing gradient problem and improves the network robust to non-linearity in the data.

**3. Pooling Layer / Downsampling layer :**

It is a non-linear down-sampling layer. Its main purpose is to reduce the number of parameters for the next convolutional layer or the output layer. The pooling layer also plays an important role to control overfitting in the network.

Some of the common pooling techniques are max pooling, average pooling, and L2-norm pooling.

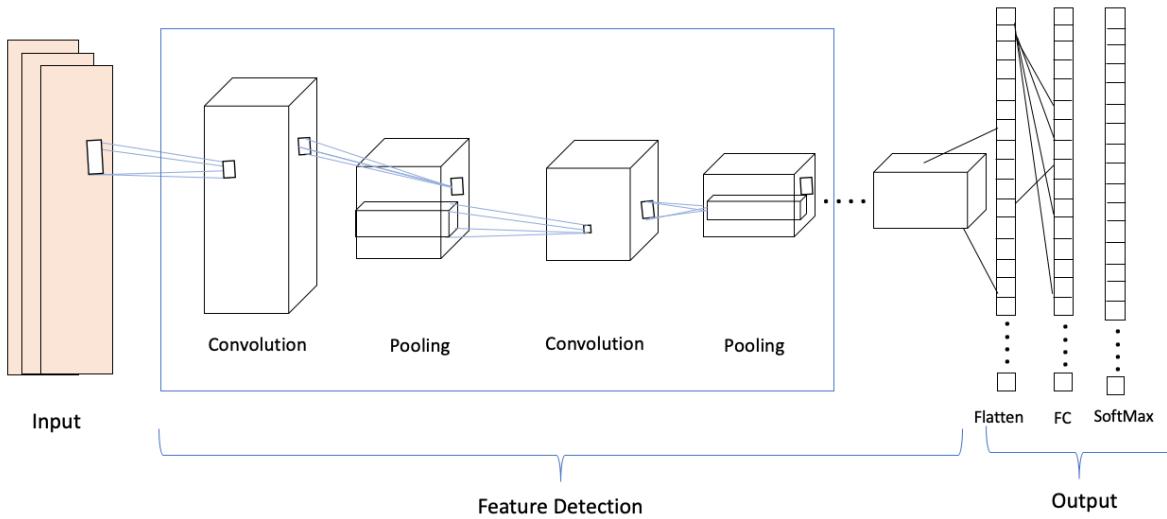


Figure 2.2.: The building blocks of convolutional neural network

#### 4. Drop-out Layers:

Drop-out [4] is the generalization technique that randomly sets off some of the neurons in the layer to the value zero during forwarding propagation.

This simple technique showed a huge boost in the performance of the CNN network. It avoids overfitting in the network. Dropout is only be applied during the training of the neural network.

#### 5. Fully Connected Layer (FC):

Following the series of convolutional and max-pooling layers, all the outputs are usually connected to the fully connected layers. The incoming neurons to the FC layers contain the high-level features and the FC layers correlate the high-level features with the suitable target class and output it.

FC layer is the usual neural network layer with weights and activation functions with all the neurons in the layer is connected with all the neurons in the next layer.

In the case of the classification problem, the FC layers use activation functions like softMax to calculate the rightful probabilities for different classes and outputs the result.

##### 2.1.3. Reasons for the rise of Deep learning

The recent increase in the performance of Deep learning algorithms are contributed to many factors and some of them are listed follows:

**Availability of a large amount of data** Because of the availability of big data. The deep learning algorithms are able to produce results that are generalized to the samples that have never seen during the training time.

**Improved computation infrastructure** The improvement in the computation infrastructure like high-performance CPU and GPUs reduced the training time of the algorithms to a large extent

**Development new deep learning concepts** Because of the above advantages, the research communities proposed many numbers of generalization concepts like dropouts, many new activation functions like ReLU, Elu and a huge number of new architectures are all the reasons for the breakthrough of deep learning in the various fields.

## 3. Literature Review

### 3.1. Classical Mesh Segmentation algorithms

Conventional mesh segmentation is the hot research topic for many decades and it has a lot of applications in computer vision and computer graphics. The mesh segmentation algorithms are broadly classified into two important approaches: geometric and semantic segmentation. The distinction is made by how the algorithm extracts the feature from the mesh. The mesh segmentation algorithm can be cast as an optimization method such that the mesh can be segmented based on some partition criterion as an optimizer.

Geometric -based approaches methods used geometric properties of the mesh as a partition criterion. Some of the common criteria are curvature of mesh, distance information, etc... Geometric approaches are widely used as the preprocessing step for other geometric based processing tasks. Most of the clustering segmentation techniques come under geometric based segmentation methods.

Semantic-based methods make segmentation labels based on human understanding and body/shape-based information. Semantic approaches use biases like shape diameter function (SDA) as a partition criterion. Not one segmentation method can solve all the problems and so it is not easy to compare the capacity of any two methods together.

In this section, some of the important mesh segmentation algorithms are discussed.

#### 3.1.1. Segmentation Types

The segmentation types can be made based on how the mesh data is used during the segmentation task. It depends on how the data is acquired for the segmentation task - such as the 3D volumetric view or 2D surface view.

The part-type segmentation method converts the 3D volumetric mesh data into different parts of semantic segments. The surface-type segmentation sees the mesh surfaces as the surface patch and makes the segmentation decision mostly using geometric properties.

#### 3.1.2. Partition Criterion

The partition criterion are basically designed using one or more important mesh attributes that can contribute an important role in the segmentation learning process. The algorithm groups the elements in the mesh together based on certain kinds of constraints that are imposed by the partition criteria.

The constraints that are used by the segmentation algorithm can be divided into the following types:

1. Cardinality Constraints: It is about the partition of segmentation. Usually, it eliminates the smaller or larger segmentation part.
2. Geometric Constraints: Constraints are based on geometric properties like a maximum area that is allowed for the sub-mesh.
3. Topological Constraints: Applies topological constraints on the sub-mesh.

The constraints are designed based on mesh attributes. The most commonly used mesh attributes by the partition criteria are curvature, symmetry, geodesic distance, convexity and much more.

#### **3.1.3. Segmentation Techniques**

Clustering techniques [5] are the most commonly used technique in the mesh segmentation algorithms. The ultimate goal of the segmentation technique is to assign each element in the mesh into a group/sub-mesh where clustering techniques are the best candidate to achieve that goal.

Some of the important classical mesh segmentation algorithms are given below :

1. Region Growing:

This algorithm [6] follows the local-greedy approach to perform segmentation. The initial step of the algorithm is to choose a seed element and builds the sub-mesh step by step using the partition criterion. The main limitation of the region growing algorithm is its inefficiency to segment the small regions. It also needs to incorporate post-processing methods to sharpen/smoothen the segmentation boundaries.

1 (a). Multiple Source Region Grow [6] : The main difference of this algorithm from the region growing is that the initial step starts with multiple source seeds and makes the increments of all seeds in parallel.

1 (c) Watershed algorithm:

This algorithm [6] also uses multiple initial seed sources and uses average geodesic distance (AGD) as the partition criteria. The aim of the algorithm incrementally makes each seed to grow until it finds local minima of the criterion function and groups the elements into watershed partitions.

2. Hierarchical Clustering

This algorithm [7] search for local minima of each seed region separately. The performance of the algorithm is hugely correlated to the selection of the initial seeds.

Sometimes these techniques gives poor results globally. Hierarchical clustering can be called a global- greedy algorithm because it considers all clusters during the merging step and not influenced only by the growing cluster. The main distinction of this algorithm from others is the merging criteria used to perform the clustering method.

### 3. Iterative Clustering:

The algorithms [8] discussed previously are all come under the non-parametric methods because of the uninitialized cluster number for the output. In the iterative clustering, the number of the resulting clusters can be parametrized and the segmentation task can be structured as an iterative problem to find the initialized number of segmentation clusters in the result. The popular clustering method that comes under this technique is the k-means algorithm.

The initial step involves the assignment of an element to one of the k -clusters and then iteratively the elements of the k-clusters are reassigned until the elements in the clusters stops changing.

Iterative clustering suffers from the convergence of the criterion function. Usually, iterative methods use an additional segmentation algorithm as a subroutine to reduce the convergence problem. Geodesic distances are the commonly used criterion in this method. But the calculation of the geodesic distances on the fly is computationally expensive.

### 4. The hybrid algorithm:

These types of methods are constructed using two or more segmentation algorithms. Most popularly iterative clustering and graph-cut techniques [7] are used. Iterative clustering performs the general clustering steps and produce the segmentation outputs But the borders of the segmentation parts are not sharply distinguishable. To make the boundary of the segmentation regions from fuzzy to sharp or smoothly distinguishable, a decomposition techniques like graph-cut are performed and thus creates the better segmentation results.

## 3.2. Segmentation of 3D Deep Learning

### 3.2.1. Introduction

Deep learning has produced impressive results on 2D data like images and reached state of the art performances on the task like classification, segmentation, image recognition and much more. But most of the objects in the real world are three-dimensional in shape. Many researchers are currently working to implement AI algorithms like convolutional neural networks on 3D data.

In this section, we will see different types of 3D data that are popular and frequently available and that has potential benefits to perform tasks like classification and segmentation. For each 3D data types, currently available deep learning algorithms that produce state-of-art or in par to state-of-art results on 3D image segmentation task are given.

### 3.2.2. 3D Data

Recently, the research attention has increased around 3D data because of its availability and its potential applications. Most commonly used 3D data in medical-related applications can be broadly classified into two types which are given below:

1. Euclidean Data: The 3D data which can be represented by global parametrization and can be described by a common system of coordinates comes under the 3D Euclidean data. The 3D Euclidean data can be visualized as grid-like data. Most of the 2D deep learning algorithm can be easily transferred so that it can use 3D euclidean data
2. Non- Euclidean Data: The 3D data which are irregular and cannot be parameterized globally comes under this group. It has a lot of application opportunities that can transform the technological landscape. Non-Euclidean 3D-data are abundant and each data type has different geometric and structural properties and a major limitation to using the convolutional neural network on the non-euclidean data is because of its irregularity and the lack of vector space structure. A lot of new convolution function and other techniques have been proposed so that deep learning algorithms can be used with non-euclidean data. Some of the best algorithms currently available for segmentation tasks are documented in the following section.

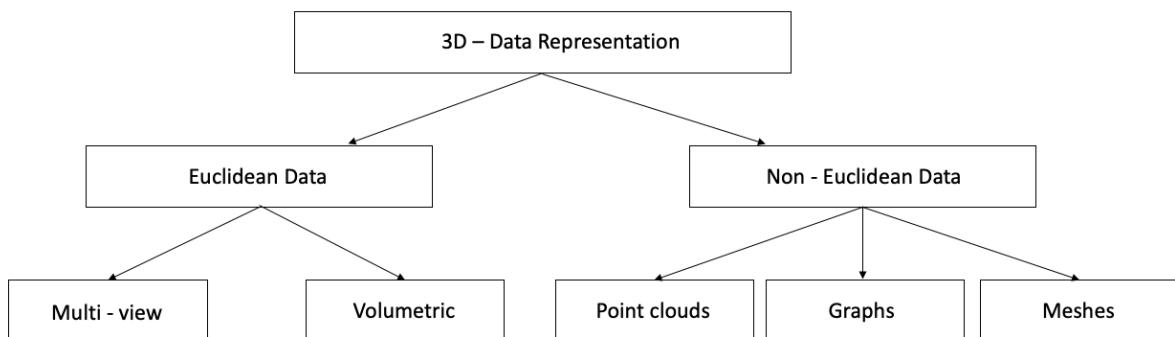


Figure 3.1.: Various types of the 3D-Data representation.

### **3.2.3. Euclidean data**

We will discuss two commonly used 3D euclidean data in the medical field and some of the powerful deep learning algorithms that works well on these 3D data. They are :

#### 1. Volumetric Data:

It is a popular and widely applied 3D data and its most advantageous feature is its regularity and grid structure in 3D space. The unit of volumetric data is called voxels which represent the distribution of geometric data in the 3D space. The main disadvantage of this data format is the usage of a large memory. Volumetric data is not suitable for when it is a necessity for high-resolution data. Applications like segmentation tasks require detailed structural information to make the segmentation decision. But voxels or octree representation does not preserve those fine-grained structural details like shapes and smoothness of the volumetric data.

ShapeNet is the first deep learning algorithm that has applied directly to 3D volumetric representation. Basically the ShapeNet introduced a new dimension in the convolutional kernel so that it can operate on volumetric data. ShapeNet has imposed many constraints and limitations that made it impossible to work on huge and high-resolution data. But it produced benchmark results on low-resolution volumetric data. The VoxSegNet [9] is the novel convolutional algorithm that performs segmentation by extracting discriminative features from volumetric data. VoxSegNet [9] uses a spatial dense extraction(SDE) so that it works on low-resolution data and to preserve the informative feature needed for the segmentation decision. The results produced from this method showed semantic consistency and high accuracy on 3D shape part segmentation. Another volumetric convolutional network called V-net [10] uses a fully convolutional network trained on MRI volumes of the prostate. The method uses a new objective function related to the dice coefficient which involves the optimization of the network during training time. The experiment results showed impressive results and new test data were tested and performance is evaluated.

Other interesting 3D deep learning publications related to volumetric segmentation are [11] , [12] and [13].

#### 2. Multi-view data

Multi-view data is the acquirement of 2D images from the different viewpoints of the 3D objects. Both volumetric and multi-view comes under euclidean data but the multi-view approach has the advantage of gathering informative features with computation efficiency. The availability of numerous 2D images helps to reduce the image distortion, noise, information loss on the acquired data. The important criteria to acquire quality 2D data from the 3D object is largely depends on the number of views taken into account. Also, many 2D deep learning paradigms can be readily applied to multi-view data without much improvisation. “Multi-View

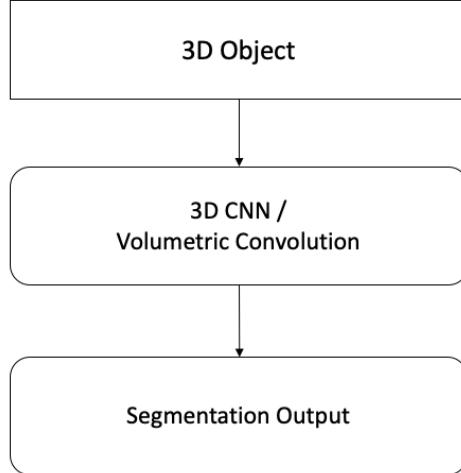


Figure 3.2.: Flow chart of volumetric segmentation.

Deep Extreme Learning Machine (MVD-ELM)” [14] was applied to the 3D sphere to acquire 20 multi-view depth images. The working of the convolutional network is very close to conventional CNN. There is also another extended version of MVD-ELM called FC- MVD-ELM. The segmentation task was performed on FC-MV-ELM then final result was project back to the 3D object using a graph-cut algorithm.

Both of the techniques provided better segmentation results than many of the other works [6] and also made a huge leap on the training and processing time.

One of the robust algorithms under this section is Multi-View CNN (MVCNN) [14]. This algorithm was applied to lung module segmentation. The network consist of 3 CNN branches that work on the input of different scale and the results of the segmentation task outperform many of the classical segmentation approaches.

### 3.2.4. Non-Euclidean data

As discussed earlier, the non-euclidean data cannot be described using a global parametrization.

There are three popular non-euclidean data. They are:

- a) Point clouds
- b) Graphs and
- c) Meshes

In the following section, each of the above-mentioned non-euclidean data will be explained briefly with an account on state-of-art publications of each data type.

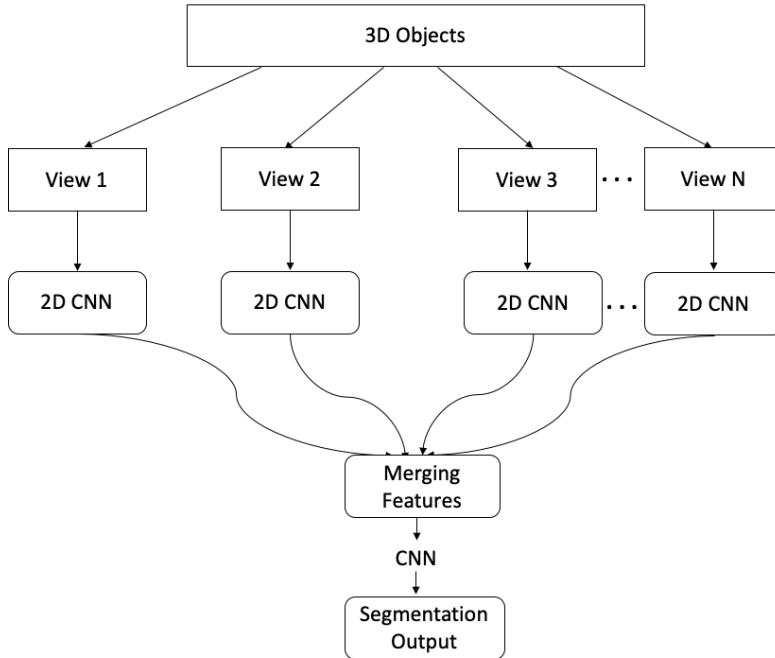


Figure 3.3.: Flow chart of multi-view segmentation.

3. Point cloud Point-cloud is just the group of point sets that approximates the structure of the 3D objects. The point clouds possess the non-euclidean nature globally however when the point clouds are sub-divided into smaller sub-sets. The sub-sets can be expressed as the euclidean data which can be parameterized (locally) and a common system of coordinates can be established.

The point cloud is the widely available non-euclidean data. It is the primary data type of signals that are generated by the technologies like imaging sensors, particle accelerators, etc...The processing of point clouds is tedious because of its lack of geometric structure. In spite of the disadvantages, point cloud data is the most research 3D data for the adaption of deep learning algorithms in the field of computer vision.

PointNet [15] is one of the earliest and powerful architectures that produced impressive results on point cloud classification and segmentation. The network is composed of three important blocks :

- a) "Spatial Transformer Network (STN)" module
- b) RNN module
- c) Simple symmetric function

All the points in the input point cloud are processed by the STN module and passed through the RNN. The RNN can learn the pattern from the data irrespective of the sequence of the input stream. Then the max-pooling is applied to the data and aggregates the output. PointNet is robust to the partial availability of data

and the perturbation changes in the input. The main disadvantage of this method is its limitation to learn the fine grain patterns of the data because of the max-pooling aggregation of data. To overcome the previously mentioned disadvantage, PointNet++ [16] is proposed. PointNet++ consists of a hierarchical neural network that recursively performs partitioning on the point cloud. Also, this method can really produce better performance even though there is a variation in the point cloud density. It is believed the hierarchical structure of the network is the main reason for its robustness.

The PointConv [17] method introduced the convolutional function that performs exclusively on point clouds. The Convolutional operation is performed by the multilayer perceptrons and kernel density functions. The convolutional operator can also be de-convolved to get back the original resolution of the point cloud. The segmentation task with this network produced state of the art performance on point cloud benchmark dataset. Most of the novelty in the point cloud convolution operator is created using kernel functions. In this paper [18], a new spherical kernel is imposed on the graph convolution. The spherical kernel produced features from the point cloud which are translated invariant and efficient with a large dataset like ShapeNet and ScanNet dataset.

ConvPoint [19] is the continuous convolutional concept which performs a convolutional function with the continuous kernels. This method is robust to varied point size and the produces state of the art performances on the part - type segmentation task.

Other interesting 3D deep learning publications related to point cloud segmentation are [20], [21] and [22].

## 4. Graphs

Graph data has a large number of applications in the current world and the geometric properties shared between meshes and graphs are similar to each other. It implies the graph network can be easily transferrable to the mesh problem. The proposed graph convolutional network can be divided into two groups based on its convolutional operator. The graph CNN can be built using a special convolution method or spatial convolution method.

Spectral filtering methods: Spectral CNN (SCNN) [23] network takes advantage of the spectral eigendecomposition technique of the graph Laplacian function to design the convolutional operator. Firstly the signals are converted into different scales using eigenbasis and then the convolutional operator is applied to the new locally scaled filter domain.

Spatial filtering methods : The CNN of this method performs the convolution

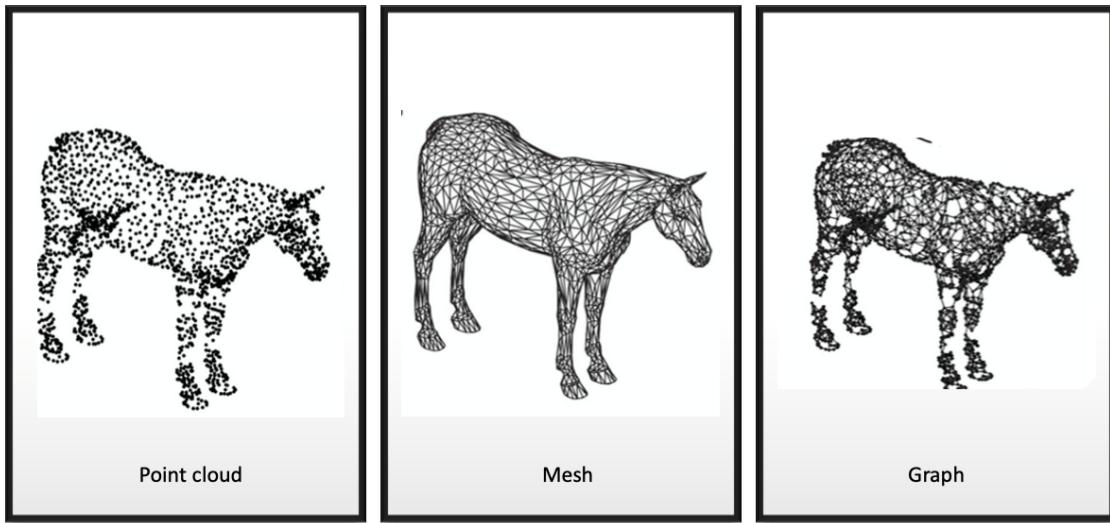


Figure 3.4.: Various types of the 3D-Data representation [6] .

operation using spatial filters like high-pass/low-pass operations. Pooling operation is usually done by graph coarsening. The publication related to spatial convolution on the non-euclidean data is very less to the spectral filtering methods.

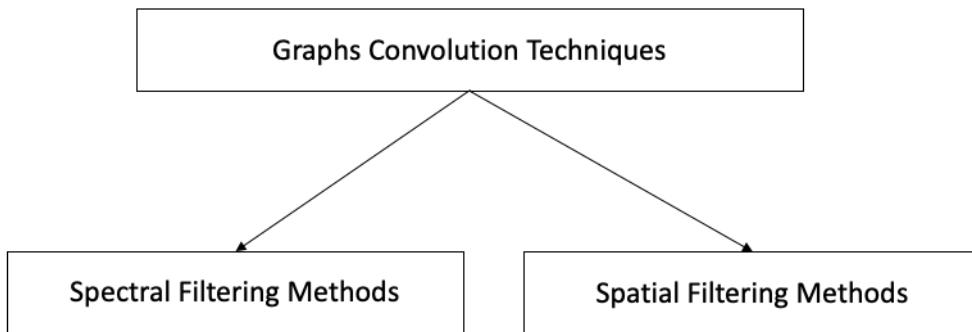


Figure 3.5.: Various types of the 3D-Data representation.

SyncSpecCNN [24] comes under the spectral-type filter method. The paper presents a novel parametrized kernels that can work on vertex function of the graph and perform convolution operation. SyncSpecCNN takes advantage of both primal and dual domains for convolution. With the proposed kernels the network can perform multi-scale analysis on different parts of the graph. The kernels can aggregate the

likely information and use that for all other of the graph even though it is dealing with different shapes. It can understand the fundamental correlation between the graph information. The architecture consists of dilated convolutional kernels that are mentioned above and followed by the spectral transformer network. The major advantage of this network is that the efficiency to perform multi-scale analysis.

Another interesting paper [25] is “Dynamic Graph CNN for Learning on Point Clouds”. It proposes a plug-and-run module called EdgeConv. EdgeConv was targeted to use the geometric relationships among points and understand the feature of the local neighborhood and also maintaining permutation invariance of the point cloud. It was shown in the paper that EdgeConv was capable to group the points and perform the segmentation task. The main advantage is that this module can be incorporated into any other point cloud architecture.

It is also important to note that there are research works that have been going on to make GCN architecture as robust as 2D CNNs. Google has proposed a network called DeepGCN [26] with the aim of making the GCN architecture with deep layers. They incorporated classical CNN techniques like residual connections, dilated convolutions into the GCN architecture and developed the network of deep 56-layers. It shows the promising results on the applications like semantic segmentation without facing vanishing gradient problem.

Other interesting 3D deep learning publications related to graph segmentation are [27], [28] and [29].

## 5. Meshes:

Similar to graph datatype, meshes also suffers from a lack of shift-invariance property. To work on the meshes, the surface structures are usually viewed as local patches and the operations like convolution are computed on it. So, meshes follow shift-invariance locally built does not hold the property globally. Geodesic CNN [30] is designed for triangular meshes. It follows the concept of expressing the local patches as local polar coordinates using patch operators. Some of the limitations of the geodesic CNN is the requirement of clean triangular meshes without irregularities, the convolutional filters that are employed may undergo arbitrary rotations which may cost the computational efficiency.

To overcome the drawbacks of rotation ambiguity of the geodesic CNN, a concept of multi-directional geodesic CNN [31] is proposed. It was particularly designed to perform the convolution well on the curvature of the meshes so that the network can share the rotation information on across the other layers and the different graph shapes.

Recent advancements in mesh convolutions are developed by the B-spline functions. B-

splines are continuous in nature and hence it can be used as continuous kernels of constant number weights and perform convolutional operators. SplineCNN [32] is the complete spatial domain filter whose size is independent of the computation time of convolution. Spline CNN showed great performance on various semantic segmentation task.

Other interesting 3D deep learning publications related to mesh segmentation are [33], [34] and [35].

## 4. Methodology

### 4.1. Introduction

The central idea of this thesis is to implement the MeshCNN network introduced in the paper “MeshCNN: A Network with an Edge “ for aneurysm dataset. Polygonal meshes can represent the 3D shapes in an efficient and robust way. In this paper, a novel convolutional neural network is implemented that can accept mesh edges as inputs. MeshCNN employed specialized convolution and pooling techniques that can be applied directly on meshes without any transformation. The network is specifically designed for triangular meshes. MeshCNN uses the edges of the mesh as the learning feature of the network like how the standard CNN uses pixels of the grid data (image) to perform the task. The main advantage of the MeshCNN is the mesh convolution and pooling operators which eliminates the need for regular and uniform representation for the neural network.

MeshCNN is flexible and can train from the input of varied sizes. It is also unaffected by the triangulation variation of the training samples. Most importantly, because of the usage of mesh convolution and mesh pooling operators employed in the network, it eliminates the need for the conversion of irregular mesh structure to regular and uniform representation which is usually needed for the convolutional computation of neural networks.

#### 4.1.1. Input Features

MeshCNN takes the edges of the input mesh sample as the feature for its training. It extracts a 5-dimensional feature for each edge and uses it for learning the task. Each edge feature consists of a 5-dimensional vector which has the following edge features as the values in it.

1. Dihedral angle
2. Two inner angle
3. Two edge-length ratios, one for each neighbor face.

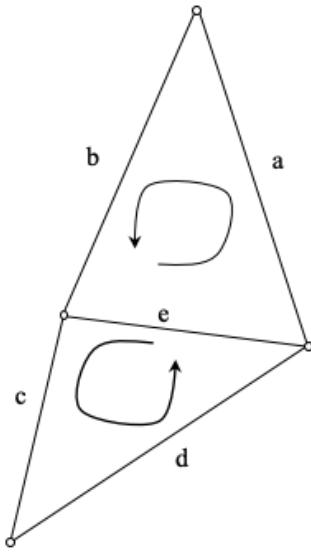


Figure 4.1.: 1-ring neighborhood of edge e.

## 4.2. MeshCNN Network

### 4.2.1. Mesh Convolution

Convolution function is performed on the edges of the mesh input. It takes advantage of the spatial support of the mesh structure. MeshCNN makes the important assumption that the input meshes are non-manifold in nature with or without boundary edges. So each edge is at most incident on the two triangular neighbors' faces. It can also be assumed that each edge has 4 or 2 neighbour edges. When It takes a 1- ring neighbor of two faces in the anti-clockwise direction as illustrated in figure 4.1. The edge e can generate two variants of the 1-ring neighborhood (a, b, c, d) and (c, d, a, b). This violates the requirement of invariant features in the receptive field for the convolution.

To perform Convolution operation on the edges, it is important that the features are similarity invariant. Hence they implemented a symmetric function that gives only the value of the relative geometric feature. Finally, the convolution operation is applied to the features generated by the symmetric function.

The definition of the classical convolution is the dot product of the kernel  $k$  and its neighborhood.

In our case, the neighborhood is the four edges So the convolution of the edge  $e$  and its four edges can be written as follows

$$e \cdot k_0 + \sum_{j=1}^4 k_j \cdot e^j \quad (4.1)$$

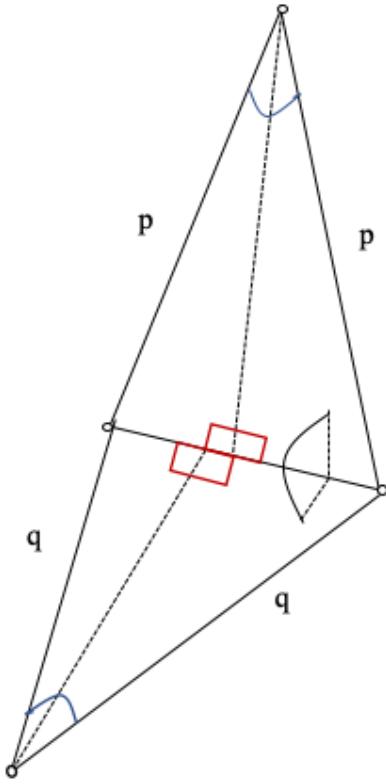


Figure 4.2.: Input features of MeshCNN network.

$J$  is the  $j^{th}$  neighbourhood of the edge  $e$ .

The four edge neighbourhood  $(e^1, e^2, e^3, e^4)$  can be  $(p, q, r, s)$  or  $(r, s, p, q)$ . To guarantee convolution invariance, the following symmetric function is employed.

$$(e^1, e^2, e^3, e^4) = (|a - c|, a + c, |b - d|, b + d) \quad (4.2)$$

The symmetric function produces new values convolutional neighbors that satisfies our assumption of similarity invariance and make the convolution function not influenced by the ordering of the neighbor edges.

After the operation of convolution, a new batch of features is generated. The number of generated features are equal to the number of kernels used on the feature.

Pooling operation is followed by the convolution, which generated new neighbors for the convolution of the next layer.

### 4.2.2. Mesh Pooling

MeshCNN uses a special pooling operation that works on irregular structures like Mesh surface. As explained in the previous section, each edge has a constant four edges convolutional neighborhood.

The basic operation of the mesh pooling is to downsample the mesh edge features and produce the resultant features only with informative edges.

Mesh pooling operator eliminates the less informative features by implementing classical mesh operation called edge collapse.

To sum up, the Mesh pooling operation can be divided into three steps:

1. Selection of the eligible edges for edge collapse
2. Performing edge collapse
3. Formulation of new adjacency neighbourhood for next conv layer

Mesh pooling learns to collapse the edges which has least information in it for learning the task.

Figure 4.3 shows the edge collapse operation performed during mesh pooling. The given five edges are collapse in to two edges and the selection of the edges to be collapsed are prioritized by the smallest norm of the edge feature which is l2 norm in our case.

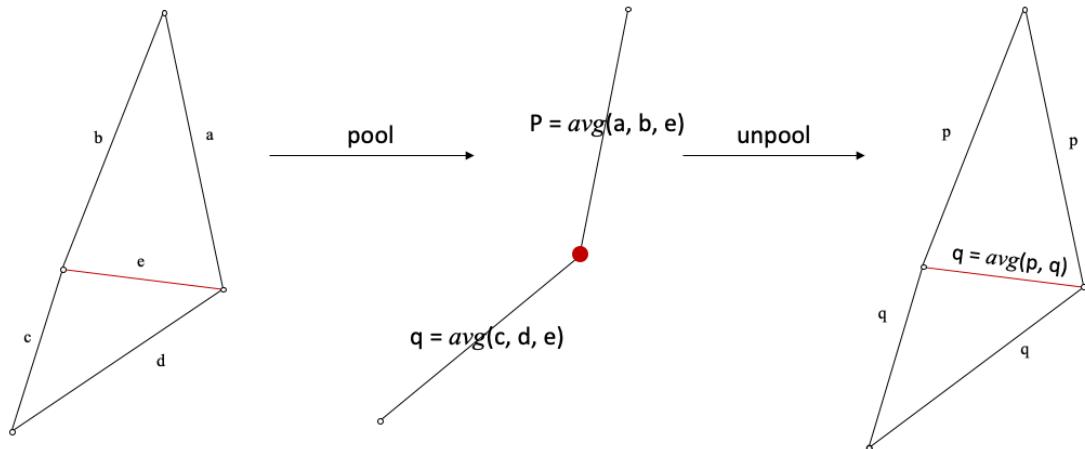


Figure 4.3.: Mesh Pooling and Unpooling Operation.

The Mesh Pooling can be mathematically defined as follows:

$$p_i = \text{avg}(a_i, b_i, e_i), \text{ and, } q_i = \text{avg}(c_i, d_i, e_i) \quad (4.3)$$

Where i is the channel index for both the triangle

It is important to be remembered that edges that would result in non-manifold edges after collapse should not be allowed to undergo edge collapse operation.

The number of resultant edges after mesh pooling can be preset by the hyperparameter. Thus the resolution of the mesh after mesh pooling can be controlled.

It is also advantageous that irrespective of the number of input edges of the mesh, the mesh pooling can produce constant edge numbers.

### 4.2.3. Mesh Un-pooling

Mesh unpooling can be considered as the inverse of the Mesh pooling. Its aim is to bring back the resolution of the mesh which was lost previously during mesh pooling. We can simply put Mesh unpooling as uncompressing or decoding step that improves the resolution as similar to the Mesh pooling which can be viewed as Compressing or encoding step that reduces the mesh resolution.

Each pooling layer is concatenated with the mesh pooling layer. So mesh pooling is performed to the same number of times as the pooling is performed.

Overall, Mesh pooling has no learning parameter. Its basic operation is to expand to feature activation in the mesh.

During Mesh pooling, the network records the edges that were compressed and hence the Mesh unpooling just gets back the resolution by using these records. Simple Mesh unpooling cannot be viewed as a learning parameter. But with the combination of the conv layer it performs unpooling and hence becomes a learnable operator.

## 4.3. Segmentation Network in MeshCNN

The obvious and easy way to implement MeshCNN for semantic segmentation task would be to use a sequence of mesh convolution layer and pooling layer with normalization and nonlinear activation layers. In order to implement pooling layer for the segmentation task, it must implement unpooling layer to bring back the resolution of the sample which was lost during the pooling steps. Hence, MeshCNN implemented a network architecture called Mesh U-Net.

### 4.3.1. Mesh U-Net

Mesh U-Net is the network architecture employed for the segmentation task in the MeshCNN paper.

It is a variant of classical U-Net. Though it is loosely connected to the U-Net features.

The network acquires U-Net shape because it performs subsequent pooling and unpooling operation during the training and results in the encoder and decoder blocks which forms the U shape.

MeshUNet also incorporated the residual connections in it. Residual connection [3] is the concepts which uses skip connections to connect the initial layers to the deep layers of the network.

Skip connections enables the deep layers to process the features in the initial layers and researches proved that it improves the training convergence of the network.

Residual connection reduces the training problem to large extent by not only enabling the signals from the initial to higher layer but also facilitating the efficient back-propagation

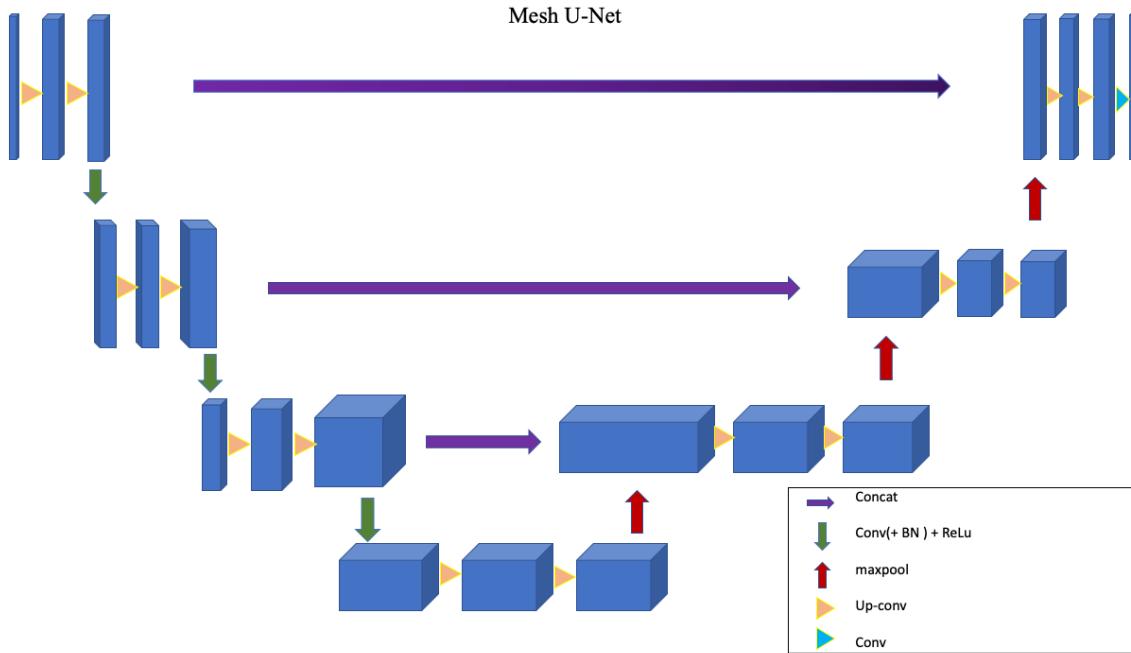


Figure 4.4.: The building blocks of the Mesh U-net.

to the initial layers during training.

Residual units [3] produces better option of making robust network with few layers and eliminate the need of large network with large number of layers. Thus it makes the network to perform task well with less number of parameter.

#### Data Preprocessing

In the paper [1], it is given that they made the edges of all the mesh samples to the same number.

They have mentioned that it is not compulsory to make all edges to the same value but they did it just to improve the time taken for the training convergence of the experiment.

Geometric mesh decimation was performed to reduce all the sample edges to uniform value.

It is mentioned that mesh resolution (input size) for the segmentation task is needed to be high compared to the mesh classification task.

The advantage is that we have the flexibility to choose the input sample resolution.

## 4.4. Augmentation

The MeshCNN incorporates few augmentation techniques in this architecture to generate augmentation samples during training.

It is important to remember that edge features are made into similarity invariant. Hence, the classical augmentation operations cannot be performed to produce new augmented samples.

So Mesh CNN used 3 types of augmentation operations to generate new mesh samples during training.

1. Anisotropic Scaling: New samples are produced by scaling the vertex location in x, y, and z ( $S_x, S_y, S_z$ ). This random scaling is performed by sampling the normal distribution.
2. Shift Vertices: As simple it sounds, Just the vertices position is shifted to a different locations on the mesh surface.
3. Random Edge flips: Chosen edges of the mesh are made to flip in the mesh structure.

## 4.5. Segmentation Experiments

### Coseg

Coseg dataset [dataset reference] consist of 3 large sets

The sets are aliens, vase and chairs of dataset values 200/300/400.

They made the train/test split to 85/15 %

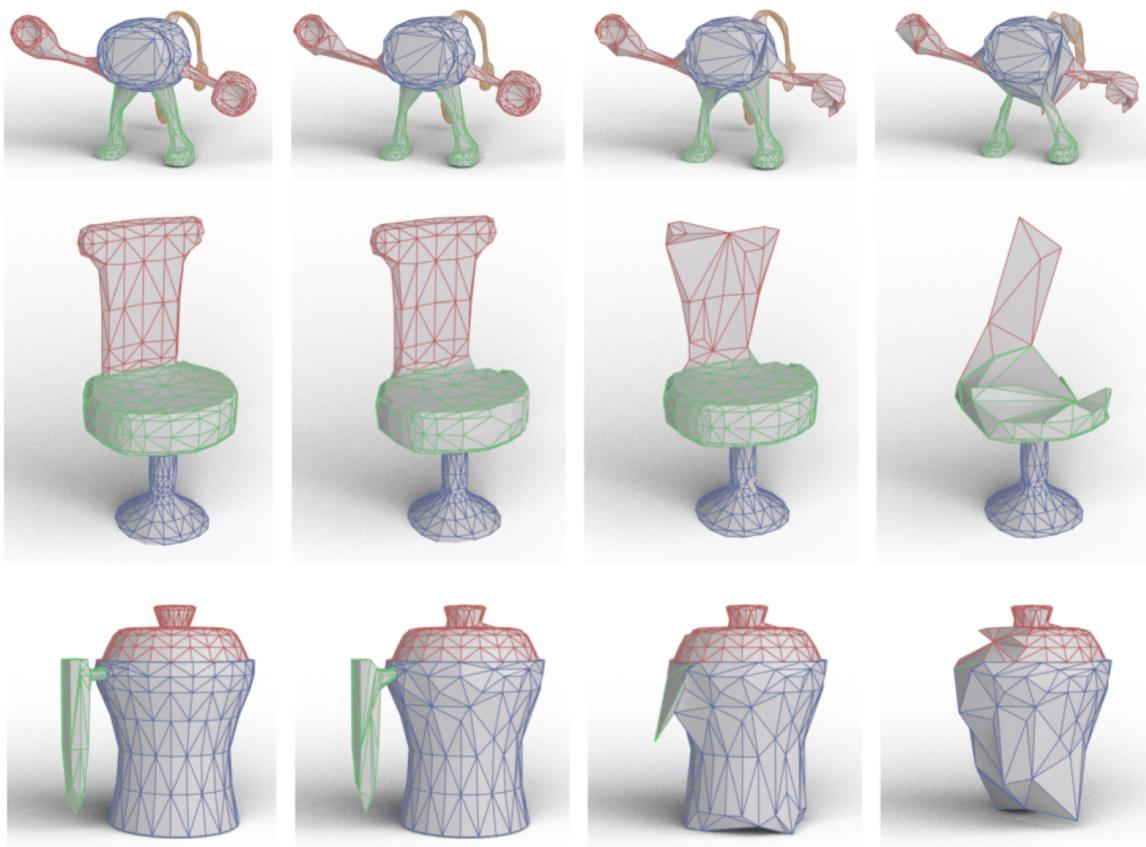


Figure 4.5.: Segmentation output of MeshCNN for the coseg dataset [1]

The experiment produce better results than any other current architecture 4.1.

**Reason :** The efficient performance of the MeshCNN on the coseg dataset is because the network is specially constructed for the Mesh inputs that give the advantage for the network to learn the Task efficient way.

To prove this, instead of structured mesh pooling the network was made to perform random pooling of the mesh which results in the comparatively bad performance of the network.

Table 4.1.: COSEG Segmentation [3]

Method	Vases	Chairs	Telealiens
MeshCNN	97.27%	99.63%	97.56%
PointNet++	91.5%	70.2%	54.4%
PointNet	94.07%	98.9%	79.1%
PointCNN	96.37%	99.31%	97.40%

### Human Segmentation Experiment

The Human segmentation Dataset [2] consists of 370 datasets where the samples are collected from the datasets.

The test set consists of 18 test samples that were borrowed from Shrec '07 dataset.

The edge-based ground truth labels are generated from the already available face-based labels.

The segmentation labels are of 8.



Figure 4.6.: Segmentation output of MeshCNN for the Human segmentation dataset [1]

### Conclusion

The novelty in the network architecture and performance superiority on various segmentation tasks compared to other sophisticated techniques intrigued us to use MeshCNN on aneurysm dataset.

Table 4.2.: Human Body Segmentation [3]

<b>Method</b>	<b>Features</b>	<b>Accuracy</b>
MeshCNN	5	92.30%
SNGC	3	91.02%
Toric Cover	26	88.00%
PointNet++	3	90.77%
DynGraphCNN	3	89.72%
GCNN	64	86.40%
MDGCNN	64	89.47%

## 5. Implementation

### 5.1. Introduction

The main part of this thesis is to generate the ground truth files for the available aneurysm dataset. To implement meshCNN algorithm on aneurysm dataset, two types of ground-truth files are needed to be generated. In this section, the segmentation parts for aneurysm meshes is explained and a step by step explanation is made about how the ground-truth labels are prepared for the pre-processed aneurysm dataset. Then the correctness of the generated labels is checked.

### 5.2. Concept of segmentation of aneurysm

The aneurysm data used in this thesis are all triangular meshes. It is necessary to rules for the segmentation parts for the aneurysm data so that we can carry out the segmentation task. The aneurysm labeling is not straight forward because of the complex structure of the aneurysm which is different for each sample.

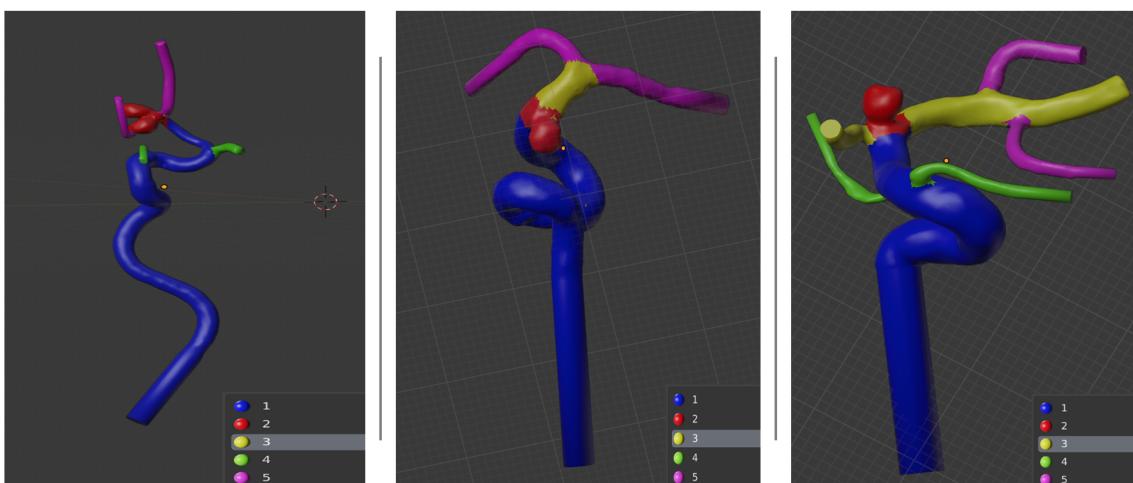


Figure 5.1.: Figure shows labels for aneurysm with at most five segments.

It is the primary requirement to make the segmentation parts applicable to all the aneurysm samples. So a concept for labeling is made which consists of 5 segmentation parts.

The structure of various different aneurysm is examined and looked for common parts that can be grouped together as a label segment. The basic sections of the aneurysm

vessels are the main vessel, branches on the main vessel, aneurysm and the branches around the aneurysm. The main interest for us is to segment the aneurysm dome from the other parts of the vessel.

Most of the aneurysm samples have the main vessel as the fundamental part. But it may have few or all the other parts in it. So the following concept was proposed to segment the aneurysm into 5 parts as shown in 5.1:

#### **1. Segment 1: Main vessel/ Parent vessel**

This segment is assigned to the parent vessel which is the inflow vessel that transports the blood to the aneurysm and the branch vessels.

#### **2. Segment 2: Aneurysm**

The label 2 is assigned to the aneurysm itself. The whole part of the aneurysm present in the mesh is added to this segment. It is important to be noted that all the samples used in the experiment have only one aneurysm in it. When the samples had more than one aneurysm it was clipped and made in to have only one aneurysm. More details about processing the mesh are given in the following section.

#### **3. Segment 3: Distal to aneurysm**

The immediate branch/branches that has arisen from the aneurysm are taken in to this segment. This segment is assigned to only the part where vessel starts from aneurysm and till the vessel branches out into many little branches. The size of the segment 3 varies from one to many vessel near to the aneurysm.

#### **4. Segment 4: Branches from parent vessel**

This segment is assigned to all the branches that have risen from the main vessel i.e segment 1. There can be one or many branches from the main vessels. All branches from the main vessel and its sub-branches are assigned to this section.

#### **5. Segment 5: Branches from segment 3.**

All the branches after the segment 3 are labelled under this segment. All the sub-branches risen from the segment 3. This segment widely varies in each sample as some have simple or complex segment 5.

### **5.3. Preparation of Dataset**

A large aneurysm dataset was given. The initial step was to remove samples that do not have good image quality and not satisfies the structure expected for our segmentation concept.

The data with following corruptions in it are removed:

1. Aneurysms with holes (non-manifolds)
2. Only head/aneurysm
3. Only parent vessels
4. Worse preprocessed one (noisy, black etc....) (Bad Quality)
5. Not well-defined mesh structure (scratchy) are removed
6. Same files (images) with differ file size
7. Repeated examples

All the samples are stored as .obj file format.

### 5.3.1. Ground-truth files

There are two segmentation text files are used for training the segmentation tasks in MeshCNN [1] . They are:

1. Eseg file (\*.eseg)
2. Seseg file (\*.seseg)

Every training sample needs its corresponding edge segmentation and soft edge segmentation file with same file name.

**Eseg file :** The edge segmentation file 5.2.

It contains a list of values of the length equal to the number of edges in the mesh sample. The value of each row contains a segmentation class that belongs to the corresponding edge of the mesh.

For example, the 1st value of the list would have the ground-truth label class for the mesh edge 1.

The Mesh CNN algorithm uses eseg file for calculating the cross-entropy loss for training the segmentation task. For our case, the row value would be any one of the 1 to 5 segmentation class values for the given aneurysm.

**Seseg file:** soft edge segmentation file 5.3.

This file only needed for calculating test accuracy. Seseg file consists of a list of values of shape. Each row corresponds to a particular edge of the given row number, any values in the columns greater then zero denotes that it is the/one of the correct segmentation label.

The soft edge segmentation ids allow the edges to have more than one correct segmentation label, which is useful in the case of boundary edges where a boundary edge can be viewed as anyone of the segments the edge shares.

Seseg file is generated by looking at the 1-ring neighbors of each of the edges and its corresponding edge segmentation class in eseg file.

46	2
47	2
48	2
49	3
50	3
51	3
52	1
53	1
54	1
55	1
56	1
57	1
58	1
59	1
60	1
61	1
62	1
63	1
64	1
65	1
66	1

Each rows belongs to the label in the range of 1 to 5

Figure 5.2.: Edge based segmentation file sample(eseg file).

Seseg file in our segmentation task consists of classes dimension.where all the edges are remeshes close to 30000 edges for each mesh sample.

### 5.3.2. Data Pre-processing

All aneurysm data are made to the same number of faces i.e. 20000 faces using blender python script.

The simplification of mesh faces to the same value is performed before training the sample because it can improve the speed of learning even with less the training parameters. But MeshCNN can be trained with the different number of edge size.

About 240 samples are cleaned and prepared for ground-truth. Many aneurysm screenshots are put together as a montage just for visualization purposes which is added in the appendix.

### 5.3.3. Vertice grouping

To make the ground-truth file, we need to assign each of the segments to the corresponding segmentation class. Vertices group option in the blender software 2.8 can be the best

815	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
816	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
817	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
818	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
819	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
820	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
821	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
822	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
823	0.0000000	0.0000000	0.2500000	0.0000000	0.7500000	
824	0.0000000	0.0000000	0.2500000	0.0000000	0.7500000	
825	0.0000000	0.0000000	0.2500000	0.0000000	0.7500000	
826	0.0000000	0.0000000	0.2500000	0.0000000	0.7500000	
827	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
828	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
829	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
830	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
831	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
832	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
833	0.0000000	0.0000000	0.2500000	0.0000000	0.7500000	
834	0.0000000	0.0000000	0.2500000	0.0000000	0.7500000	
835	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
836	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
837	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
838	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
839	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
840	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
841	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
842	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
843	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
844	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
845	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
846	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
847	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
848	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
849	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
850	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
851	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
852	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
853	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000

Figure 5.3.: Face based segmentation file sample (Seseg file).

option to get it done. Basically, we can certain the vertices of a mesh into a group and assign a name for it in the blender software.

So we can take advantage of this option and transfer the information from the vertice grouping of the mesh to the python environment using the python interface of the blender 2.8.

Steps involved in vertice grouping are:

1. Selecting all the vertices that correspond to the segmentation class (say, 1)
2. Assign the selected vertices to the group with some name (say 1)
3. Repeat step 1 and 2 for all the segmentation class
4. Run the python code in the blender python interface to transfer the information to a text file.

The screenshot of blender software is given in figure 5.4. Where in the right corner it can be seen that there is group with names 1 to 5 which is created to assign corresponding segmentation class vertices to it.

And also figure 5.5 highlights the vertices of each of the 5 segmentation classes of an aneurysm mesh.

## 5. Implementation



Figure 5.4.: Vertice grouping of aneurysm mesh sample in Blender software

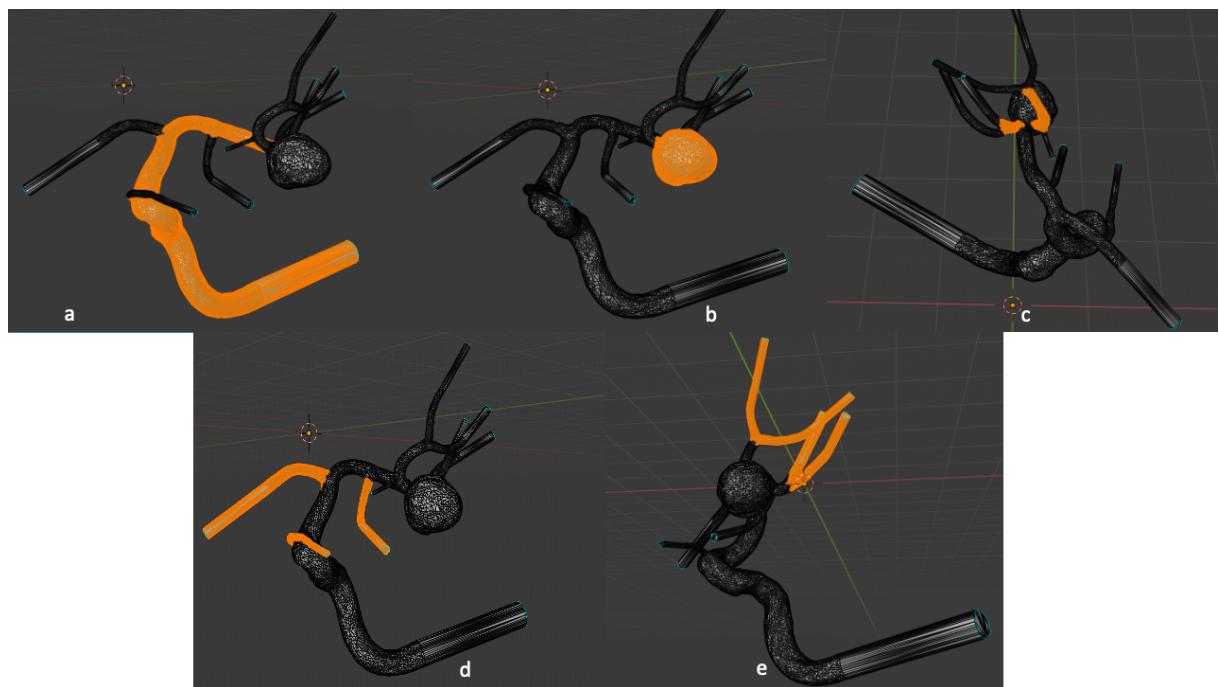


Figure 5.5.: Vertices labels are grouping for each of the 5 segments

Generation of eseg and seseg file

Using the text file generated from the vertex grouping the edge segmentation(eseg) and soft edge segmentation (seseg) file is generated using a python script.

#### **5.3.4. Workflow of ground-truth generation**

##### **Workflow of the ground-truth generation of single aneurysm data**

To sum up, the following steps are followed for each aneurysm sample to generate a ground-truth segmentation file.

1. Make the aneurysm mesh have 20,000 faces.
2. For each mesh, Vertices of the same segmentation classes are grouped together (using blender vertices-group option).
3. Run the python script to generate eseg files.
4. Run the python script to generate seseg file.

##### **Workflow of the ground-truth generation of multiple aneurysm data**

1. Cut all the secondary aneurysms except the main one using blender software 2.8.
2. Make the data to perfect triangular mesh in the blender.
3. Upsample the mesh to have 20,000 faces.
4. Perform the vertices grouping in the blender and export the information to a text file.
5. Extract new obj files, edge file, eseg and seseg files using the saved data from above (4) step.

##### **Checking the correctness of generated eseg labels**

The correctness accuracy of the generated ground-truth segmentation file is checked and the segmentation figure is generated and visualized in python.

It can be confirmed from figure 5.6 that all the segments are correct to its corresponding segmentation class. The segmented meshes denotes red for the main vessel(segment 1), purple for the aneurysm (segment 2), yellow for the branch from aneurysm (segment 3), blue for the branches from the main vessel (segment 4) and rose for the secondary branches from the segment 3 (segment 5).

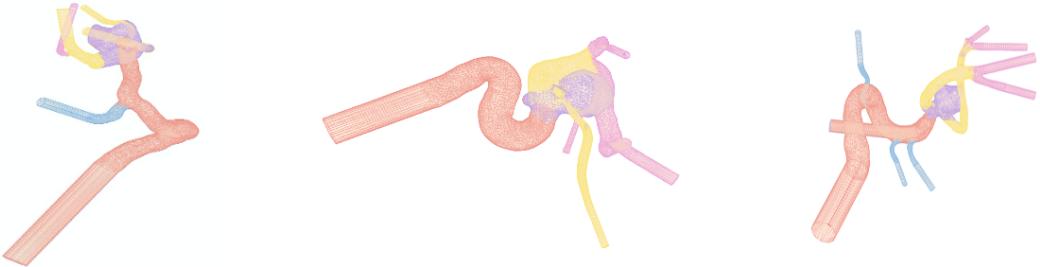


Figure 5.6.: Correctness of manual labelling is checked and the above figure confirm its correctness

### Total number of segmented samples

Around 240 aneurysm samples are vertices are grouped in the blender. But there are some errors occurred during eseg and seseg files because of the poor tessellation of some meshes.

Total number of training and test samples used can be found in the experiment and results section.

## 5.4. Augmentation of Aneurysm

### 5.4.1. Introduction

Many pieces of research and deep learning training results proved that the number of training samples largely influence the learning and performance of the deep learning algorithms. But It is always not possible to gather a large amounts of data for training purposes. Especially, the number of data samples available in the medical field is extremely limited compared to other fields. This is the main challenge that limits the speed of deep learning algorithm implementation in the medical application.

One of the main practices that are common to increase the number of training samples is the use of augmentation techniques. Many data augmentation techniques have been proliferated in the past few years in the case of 2D deep learning. But, the augmentation techniques used in 2D deep learning cannot be used for geometric deep learning and the number of augmentation currently available for 3D non-euclidean data is very less. Even the currently available techniques are not much sophisticated for a large improvement in

training performance.

To increase the training data, simple methods are formulated to produce new augmented training samples.

### 5.4.2. Steps of Augmentation method

The concept of the augmentation method is to merge two different parts of the two mesh samples together using blender tools.

For example, the aneurysm part with/without branches of the mesh is merged with the main vessel using the boolean operator of the blender.

Pipeline:

1. Bisect the aneurysm part of the Data A
2. Bisect the main vessel of the Data B
3. Join and align the aneurysm and main vessel together
4. Apply modifier (Boolean) on the joined meshes
5. Save and export the new augmented mesh.Obj file

Figure 5.7 shows the augmentation steps, where two bisected meshes are brought together and the final image shows the new augmented mesh.

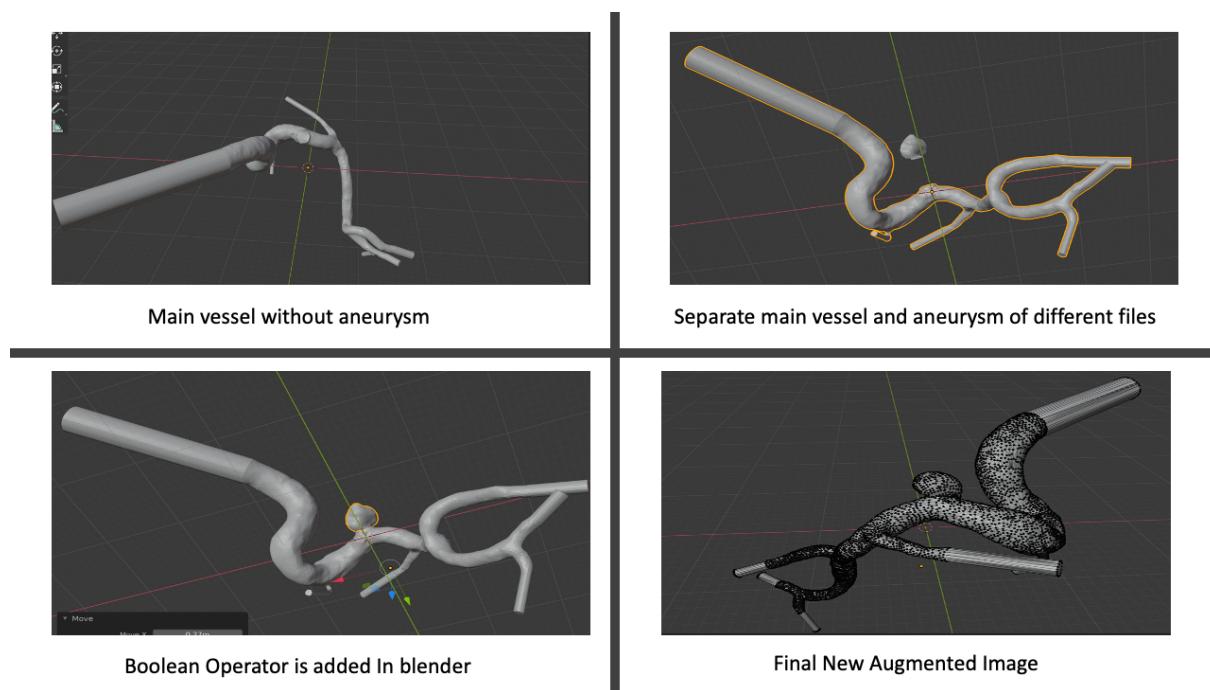


Figure 5.7.: Steps involved in manual augmentation

### **Thickness/size change of the mesh during augmentation (if required)**

The thickness of the mesh can be easily increased or decreased using blender tools easily. The change of thickness is necessary during most of the augmentation. Because not always two of the meshes to be augmented will have the same mesh diameter.

But this problem can be easily solved by just selecting the target mesh in the edit mode of the blender and click ALT+S. Then you are increase or decrease the size using mouse manipulation.

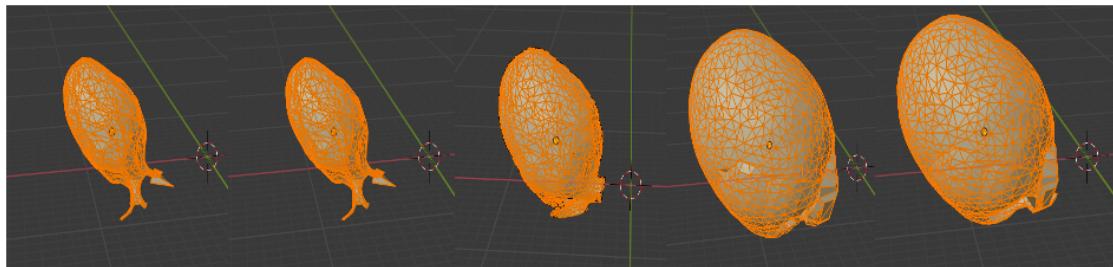


Figure 5.8.: Change in size of clipped aneurysm sample

The increase of mesh size using this technique can be viewed in figure 5.8.

#### **5.4.3. Cleaning of the new augmented file in MeshLab**

After generating a new augmented data sample, it has to clean and bad vertices in the mesh needed to be removed. Because the MeshCNN needs clean, manifold, triangular meshes possibly with boundary edges.

It is important to remove the bad geometry of the mesh before performing the ground truth generation.

Some of the steps performed to clean the augmented data are given below:

1. Remove loose vertices and clean the data
2. Clean non-manifold vertices and edges
3. Make into a pure triangular mesh

Figure 5.9 shows two examples of the augmented data that has been generated manually.

#### **5.4.4. Conclusion**

There are about 10 augmentation samples are generated using this method and the ground-truth files are generated. It has to be noted that 3D deep learning needs new and better augmentation techniques in the future.

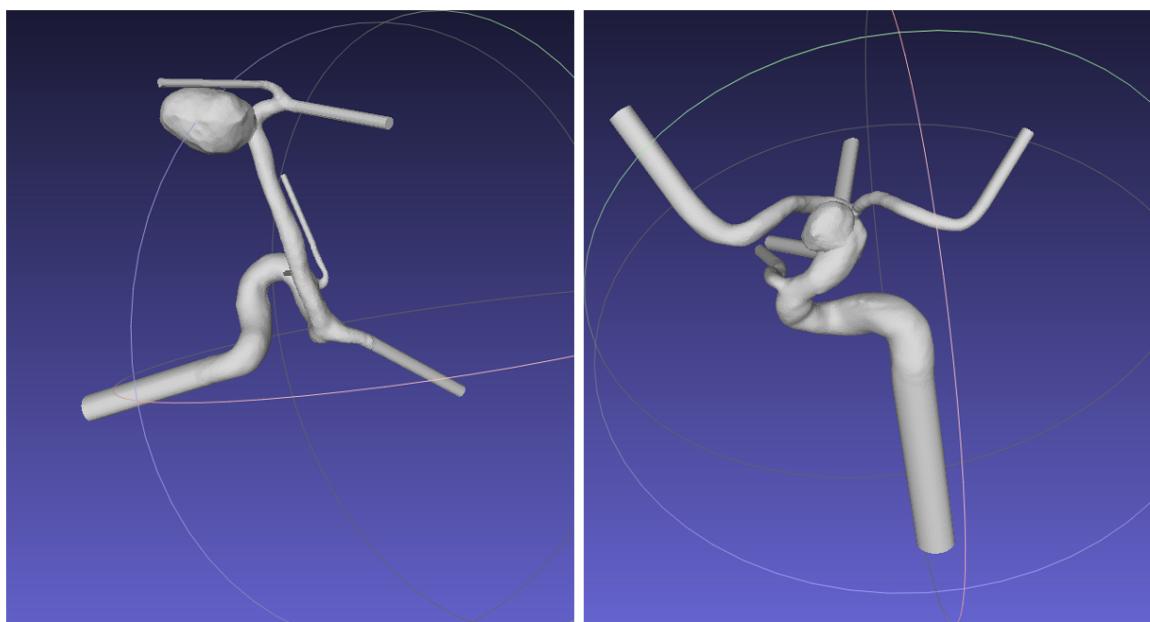


Figure 5.9.: New augmented aneurysm samples.

## 6. Experiments and results

This section consists of the information about all the experiments that are performed on MeshCNN network with aneurysm dataset. Firstly the metrics that are used by the MeshCNN for the calculation of training loss and test accuracy are explained. Then, the list of hyperparameters of the MeshCNN and its default values are added.

Finally, for each experiment, the values of the hyperparameters that are different from the default values are tabulated then, the corresponding training loss and test accuracy are also visualized and the segmentation output of the test set meshes are also given.

### 6.1. Metrics

The training loss and test accuracy of each of the experiments are given.

**Training loss:** MeshCNN uses cross-entropy loss as the training loss of the network. The cross-entropy loss measures the closeness of the output prediction and the ground-truth and gives out a loss value. The network can be viewed as learning properly when the value of the log loss curve decreases as the training continues. The training loss curve also gives information about the overfitting.

**Test accuracy:** MeshCNN uses the special information of the ground-truth to calculate the test accuracy. It uses the soft-edge information of the test samples to calculate the test accuracy of the network. More information about the soft-edge label information is given in the implementation chapter.

### 6.2. Hyperparameters list

The list of all hyperparameters of the meshCNN network is listed below. There are some of the hyperparameters which influence the learning of the training curve more than, the other hyperparameters. Input edges define the numbers of edges of the mesh sample. All the meshes of the aneurysm dataset are made into 30000 edges. The pooling layer and learning rate are the hyperparameters that have to optimize with the learning of the new dataset. Many experiments are conducted to see how the change in the pooling layer values and the learning rate values influence the learning of the network. It is also important to remember that MeshCNN uses various augmentation techniques in it. There are three augmentation techniques :

1. Vertices scaling

2. Edge flip
3. Vertices sliding

But there is no reference about the robustness of these augmentation techniques with the learning of the network in the MeshCNN paper. So, the hyperparameter values of these augmentation techniques are also varied to see the performance correlation with it.

Table 6.1.: Hyperparameters list

Hyperparameter	Definition	Values used
Batch size	input batch size	12
Architecture	network type	meshunet
Residual blocks	number of res blocks	3
Layers	number between fc and nclasses	100
Convolutional layer	convolutional filters in each layer	[32, 64, 128, 256]
Pooling layer	Residual pooling	[1800, 1350, 600]
Normalization	normalization of layer weights.	batch
Number of groups	number of groups for group normalization	16
Weight Initialization	network weight initialization.	normal
Initialization (gain)	scaling factor for normal orthogonal	0.02
Number of threads	number threads for loading data	3
Serial Batches	if true takes meshes in order else randomly	False
Iteration Decay	linearly decay learning rate to zero	2000
Adam Parameter	momentum term of Adam	0.9
Learning rate	initial learning rate for Adam	0.001
Learning rate Policy	learning rate policy	lambda
Number of Augmentation	number of augmentation files	20
Vertices scaling	non-uniformly scale the mesh along axis	False
Vertices sliding	percent of vertices which will be shifted	0.2
Edge flip	percent of edges to randomly flip	0

### 6.3. Human Segmentation Experiment

This is the preliminary experiment to see how the algorithm performs when there is a difference in the input edge size. To check if the algorithm is robust to perform the segmentation task with varied mesh sizes. The experiment was conducted on the human segmentation dataset [2] .

There are 20 new upsampled human segmentation meshes of range 3000 - 5000. Then the new meshes are added to the dataset and the experiment was conducted. Hyperparameters are kept to the same values given in the paper. The MeshCNN learns the task well with the varied input size and also the test sample of varied size is given in the figure 6.1.

This result gave the motivation that the algorithm can be a good candidate to perform segmentation task for the aneurysm dataset.

Table 6.2.: Human segmentation

Hyperparameter	Value
Batch size	12
Edge flip (flip edges)	0
Vertices sliding (slide verts)	0.2
Convolutional layer (ncf)	[32, 64, 128, 256]
Pooling layer (pool res)	[1800, 1350, 600]
Residual blocks (Resblocks)	3
Input edges	2280

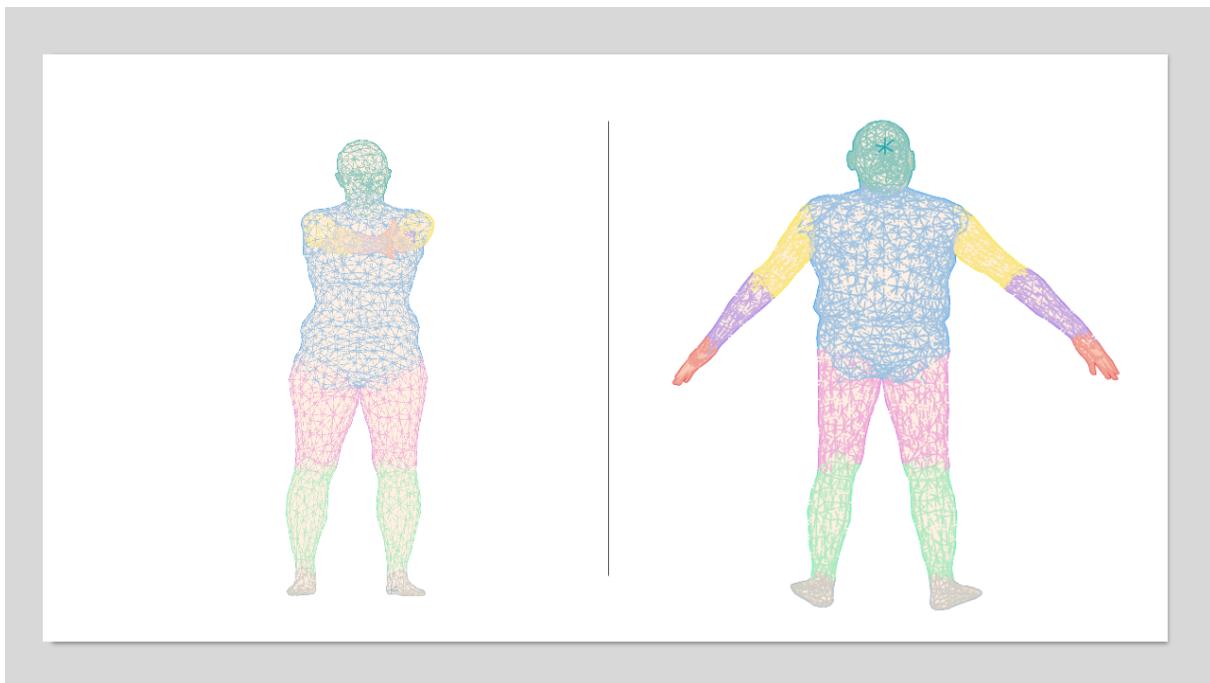


Figure 6.1.: Test result of large Human segmentation test sample

## 6.4. Aneurysm Experiment

Table 6.3.: Train/Test set

Train set	Test set
182	8

### 6.4.1. Experiment 1

This is the first experiment conducted on the aneurysm dataset. It was unclear about the range of pooling layers that would work on an aneurysm dataset. So pooling layers are not reduced very high here. The training loss and test accuracy curves were showing incremental improvements 7.2. The output of the test set can be seen in figure 6.3.

## 6. Experiments and results

---

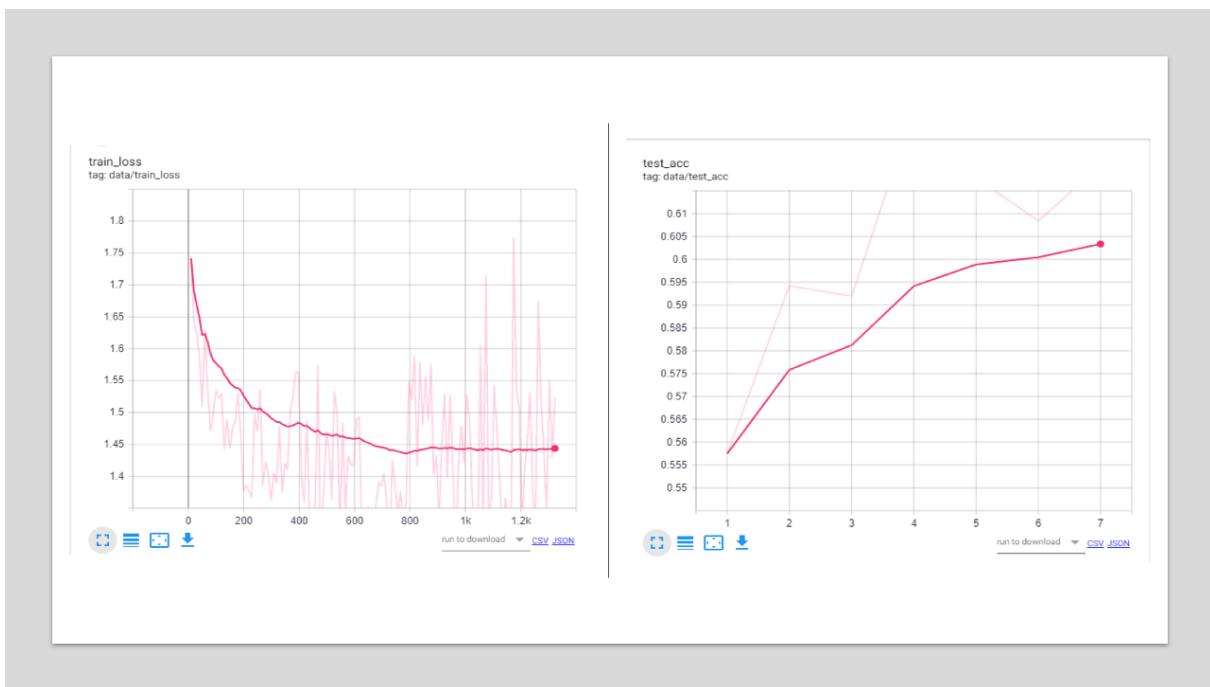


Figure 6.2.: Training loss curve (left) and test accuracy (right) of Experiment 1

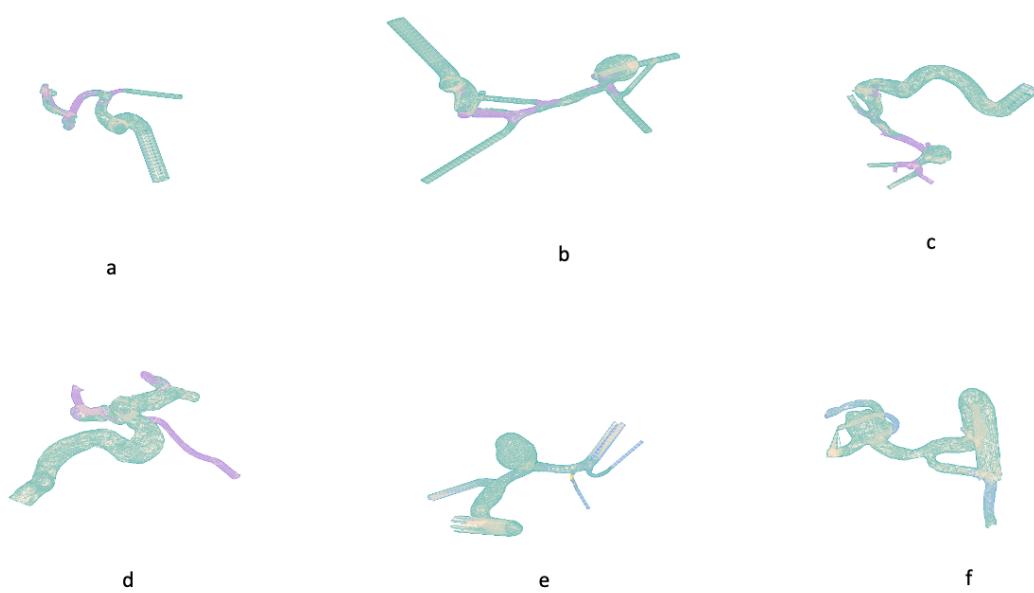


Figure 6.3.: Test set Aneurysm samples of experiment 1

Table 6.4.: Experiment 1

Hyperparameter	Value
Batch size	2
Edge flip (flip edges)	0.1
Vertices sliding (slide verts)	0.2
Convolutional layer (ncf)	[32, 64, 128, 256]
Pooling layer (pool res)	[28000, 25500, 19500]
Residual blocks (Resblocks)	3

#### 6.4.2. Experiment 2

In this experiment, the pooling layers range was highly reduced from the first to the third layer to see how it affects the performance. The training loss and test accuracy curve of this experiment can be seen in figure 6.4. The output of the test set aneurysm samples is added in the figure 6.5.

Table 6.5.: Experiment 2

Hyperparameter	Value
Batch size	2
Vertices sliding (slide_verts)	0.1
Learning rate	0.001
Convolutional layer (ncf)	[32, 64, 128, 256]
Pooling layer (pool res)	[25500, 12500, 6500]
Residual blocks (Resblocks)	3



Figure 6.4.: Training loss curve (left) and test accuracy (right) of Experiment 2

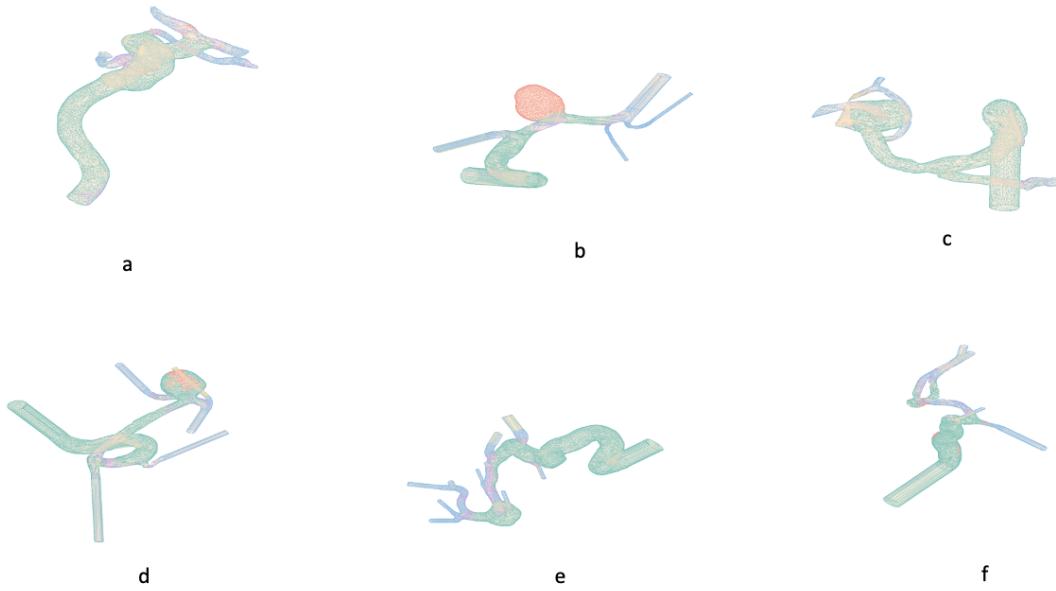


Figure 6.5.: Test set Aneurysm samples of experiment 2

#### 6.4.3. Experiment 3

In this experiment, the number of residual blocks is improved and also the value of the learning rate is reduced from the default value of 0.01. And the performance of this experiment can be viewed in the training loss and test accuracy curve 6.6. The test set samples are given in figure 6.7.

Table 6.6.: Experiment 3

Hyperparameter	Value
Batch size	2
Edge flip (flip edges)	0
Vertices sliding (slide_verts)	0.1
Learning rate	0.0005
Convolutional layer (ncf)	[32, 64, 128, 256]
Pooling layer (pool res)	[24500, 12500, 6500]
Residual blocks (Resblocks)	4

#### 6.4.4. Experiment 4

This experiment has the same hyperparameters except for vertex scaling. The augmented vert scaling is introduced to see how this method influences the learning of the network. The training loss and test accuracy curve are given in 6.8. Some of the test samples can be viewed in figure 6.9.

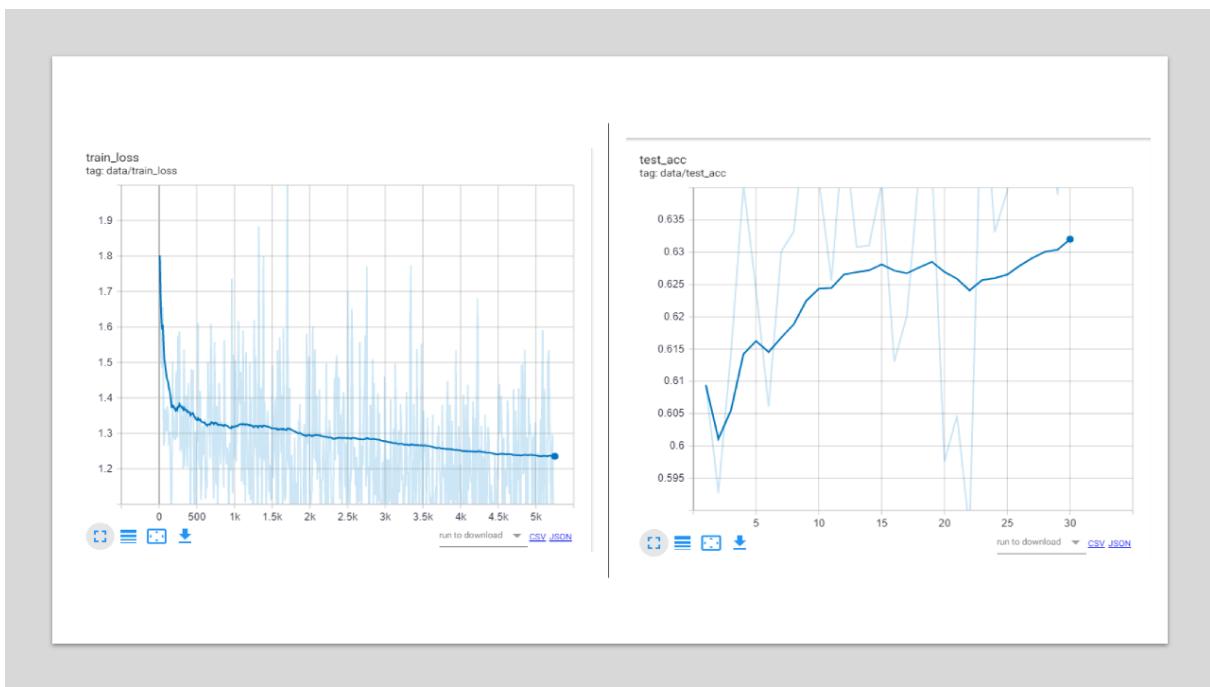


Figure 6.6.: Training loss curve (left) and test accuracy (right) of Experiment 3

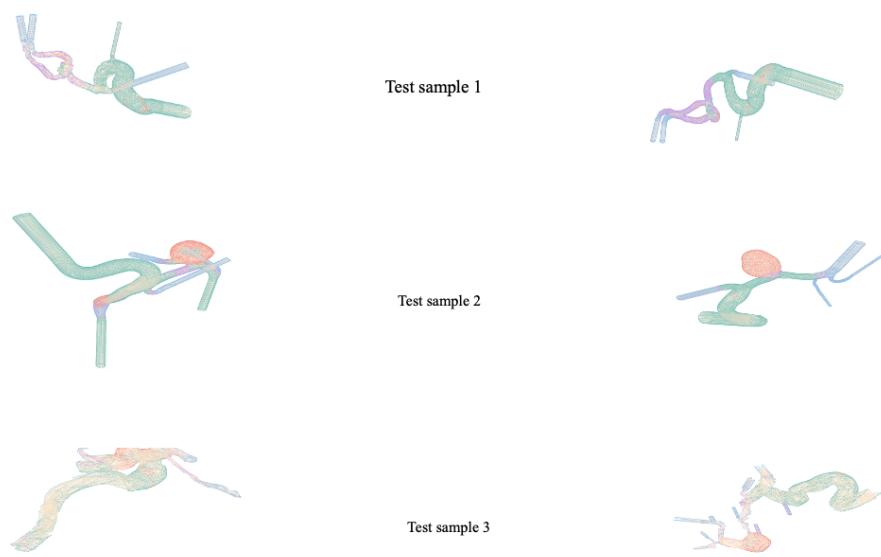


Figure 6.7.: Test set Aneurysm samples of experiment 3

## 6. Experiments and results

---

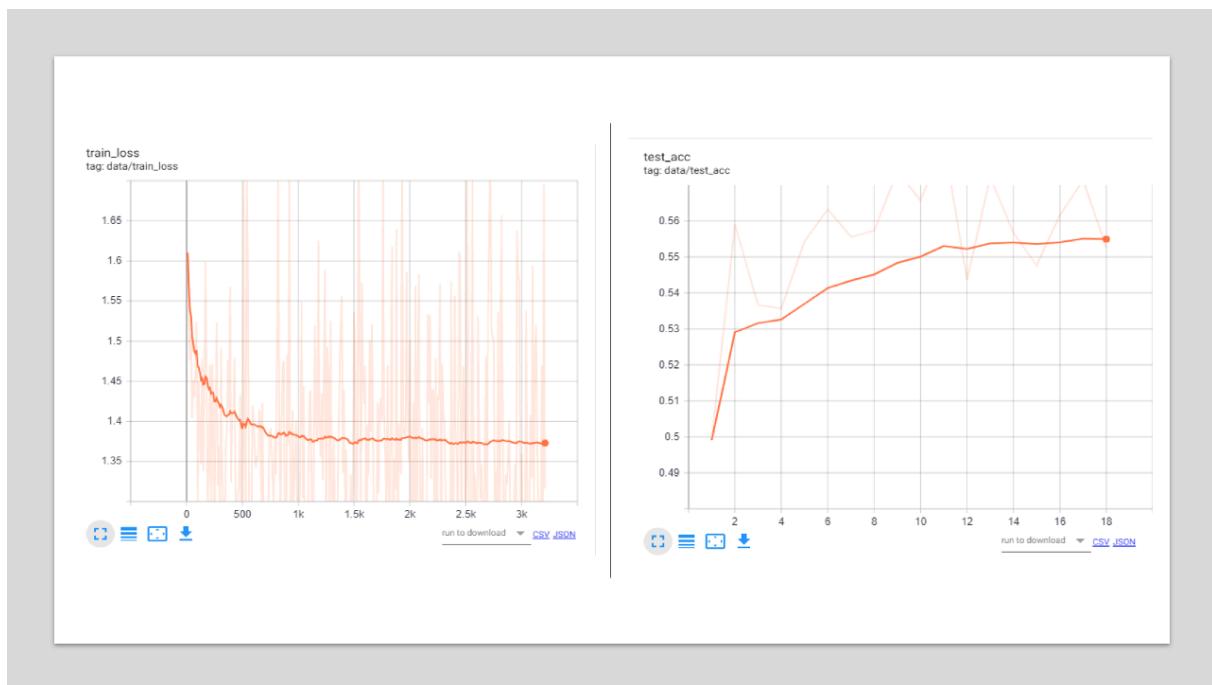


Figure 6.8.: Training loss curve (left) and test accuracy (right) of Experiment 4

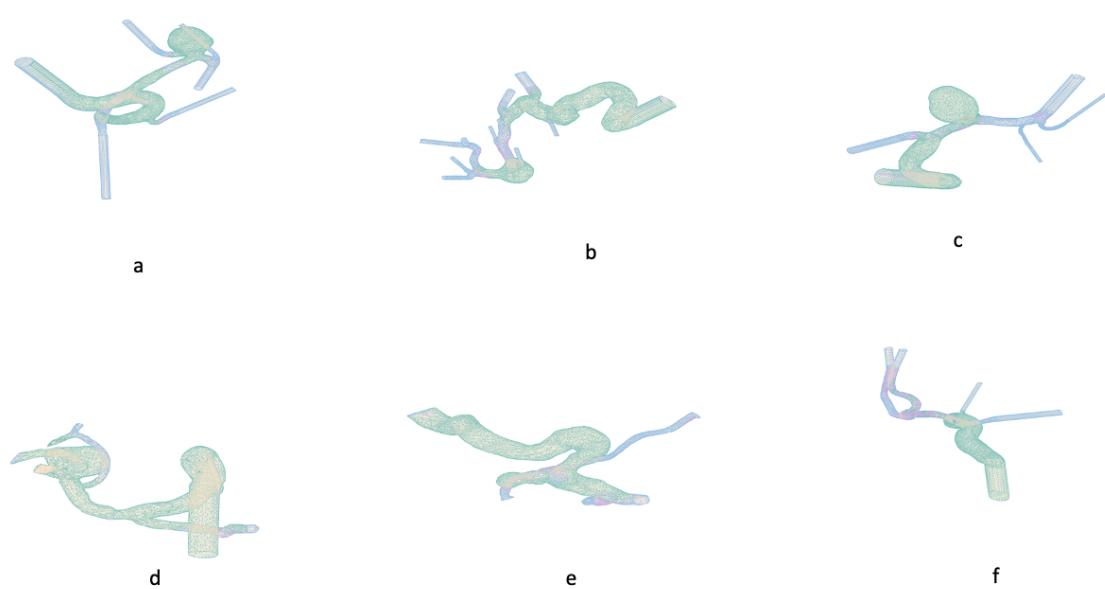


Figure 6.9.: Test set Aneurysm samples of experiment 4

Table 6.7.: Experiment 4

Hyperparameter	Value
Batch size	2
Edge flip (flip edges)	0
Vertices sliding (slide_verts)	0
Vertices scaling (scale_verts)	True
Learning rate	0.001
Convolutional layer (ncf)	[32,64, 128, 256]
Pooling layer (pool res)	[24500, 12500, 6500]
Residual blocks (Resblocks)	3

#### 6.4.5. Experiment 5

This experiment is as same as previous one except the pooling layer values are changed to check the correlation between the vertices scaling augmentation method and the pooling layer and the training and test curves are recorded which can be seen in the figure 6.10. The output of some of the test set samples are given here 6.11.

Table 6.8.: Experiment 5

Hyperparameter	Value
Batch size	2
Edge flip (flip edges)	0
Vertices sliding (slide_verts)	0
Vertices scaling (scale_verts)	True
Learning rate	0.001
Convolutional layer (ncf)	[32,64, 128, 256]
Pooling layer (pool res)	[28000, 25500, 19500]
Residual blocks (Resblocks)	3

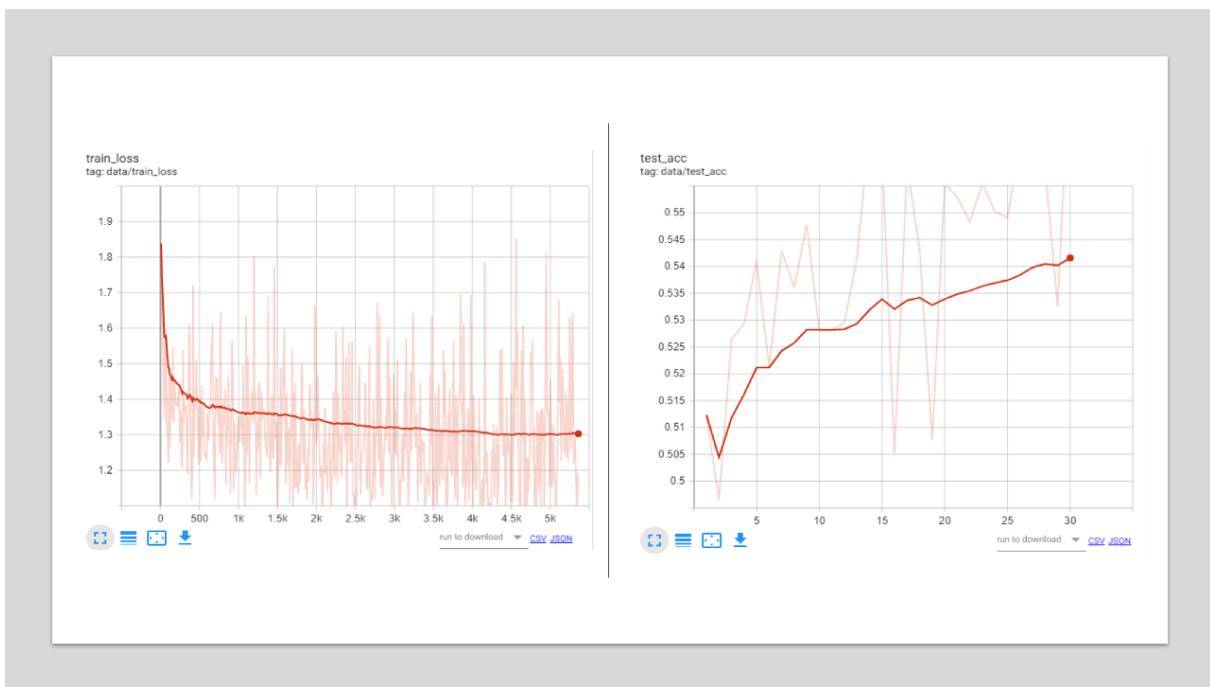


Figure 6.10.: Training loss curve (left) and test accuracy (right) of Experiment 5

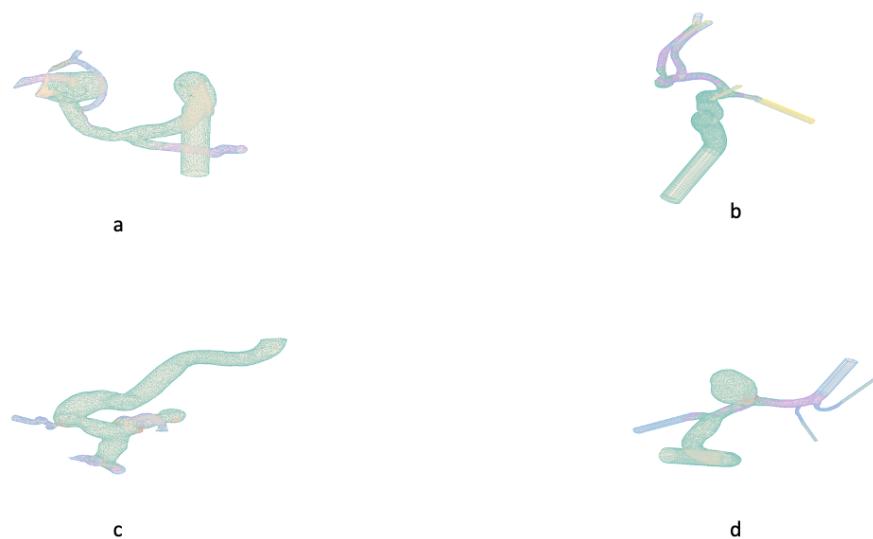


Figure 6.11.: Test set aneurysm samples of experiment 5

## **7. Dicussion**

This chapter is devoted to the discussion of the results of the experiments. In the first section, the challenges face to implement the aneurysm dataset are discussed followed by the section that contains about the discussion about the various experiments that has performed.

### **7.1. Challenges and solutions**

There are various challenges that have occurred during the thesis implementation. There are issues that have happened during the ground-truth generation, manual augmentation preparation and experiment tasks. These three challenges are explained briefly in the following section.

#### **7.1.1. Ground-truth data generation**

The ground-truth concept and file format of the MeshCNN is novel. It was unclear about the right process to perform the ground-truth generation. The generation of edge-based labeling from the vertice grouping information exported from the blender was a crucial process. The vertice information in the obj file and the vertice values in exported with the vertice-based label values are different. Then finally the problem was found out and the pipeline for the edge-based label generation for the aneurysm mesh sample was prepared. Manual Augmentation During the preparation of the manual augmentation pipeline, various errors have occurred. The errors are related to the structure of the augmented mesh. The MeshCNN accepts the inputs only made of manifold, triangular meshes possibly with boundary edges. When the mesh geometry is bad or the triangular structure is not regular it caused the error during the training of the network. There is various manual correction are needed to perform so that the output of the manual augmented sample to be clean and geometrically well structured. To make the mesh augmented mesh perfectly triangular and manifolds, various options in the MeshLab and Blender are used.

#### **7.1.2. Hyperparameter tuning and training time**

The training of MeshCNN with the aneurysm dataset is the main challenge of the project. After the generation of ground-truth for the whole aneurysm dataset. There were many errors that hinder to conduct the training of the aneurysm segmentation task. The

MeshCNN threw out a lot of errors when the mesh geometry is poor. In order to find out which meshes caused the problem and to understand how it can be eliminated took a lot of time. After cleaning the aneurysm dataset, the experiments were conducted for various hyperparameters. The aneurysm meshes are very large in size (edges of 30000) compared to the edges of the human segmentation meshes (2280 edges) used in the MeshCNN. The training time for the human segmentation dataset itself was too long (several days). As the mesh size of our dataset was large. All the experiments are conducted with the Geforce GPU. Still, our network took a minimum of 6 days to perform 5,000 epochs for a set of hyperparameters. Because of the long training time and many hyperparameters in the experiment. There was a very limited chance to find analyze hyperparameter tuning.

## 7.2. Interpretation of results

### 7.2.1. Human segmentation experiment

The preliminary experiment was conducted on the human segmentation dataset to understand the robustness of MeshCNN against the dataset with the varied input data samples. About 20 samples are created with the input edges of range 5000 - 6000. The MeshCNN didn't throw out any errors or any drastic change during the learning performance of the task. This showed that the MeshCNN was capable of doing decent learning on the samples with different sizes. After this experiment, the aneurysm dataset was prepared to carry out the segmentation task on it.

### 7.2.2. Aneurysm segmentation experiment

Because of the large size of the aneurysm samples. Each training epoch can be performed maximum only of batch size 2. The pooling hyperparameter is an important parameter that has to be tuned to perform the segmentation task. The pooling layer is crucial because when the pooling is overdone, there is a possibility of losing valuable geometric information from the mesh that is important for the network to learn and perform the segmentation task. The MeshCNN network also has three augmentation techniques in it. But it was unclear how much it influences in the performance improvement of the network. By considering these insights, around 5 experiments are performed for various hyperparameters values.

#### Experiment 1

In this experiment, three Pooling layers are given the values of [28000, 25500, 19500]. The training values and test accuracy curve of this experiment showed incremental improvement. Though there is slight evidence that the network is learning properly because of the linear decrease in the training loss. The training time was very long. The training loss was not

very strictly incremental. There is high noise in the learning curve of this experiment. The comparison of some of the output of the test set is visualized parallelly with its own ground-truth samples. It can be seen that network segments the maximum parts of the test samples to the main vessel. It shows the network is under-fitted and requires more training for the improvement of the network.

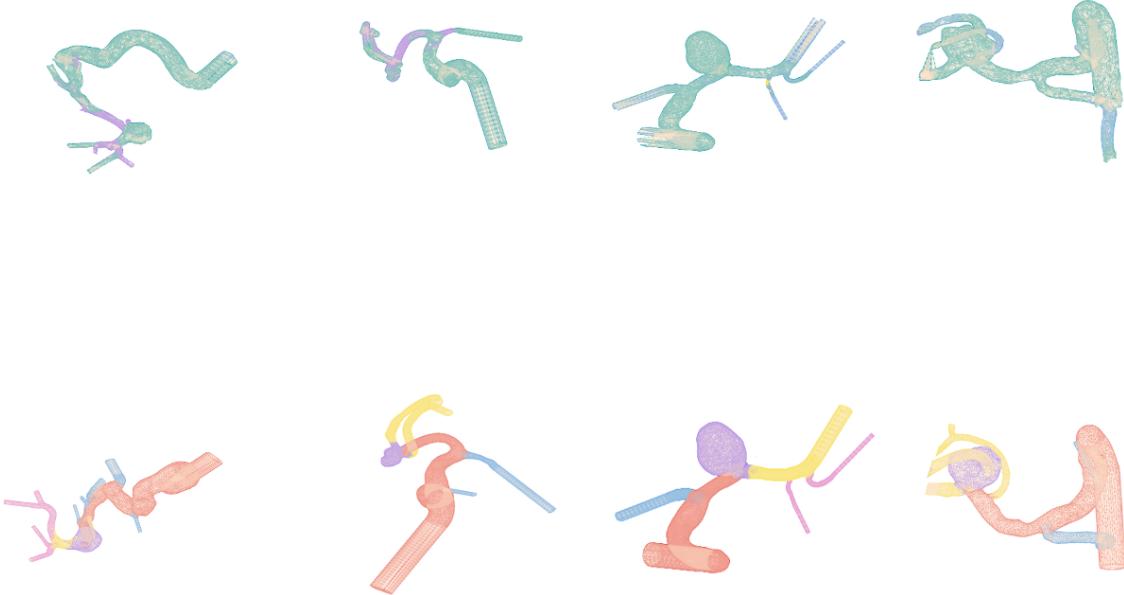


Figure 7.1.: Comparison of test samples (top row) and ground-truth (bottom row) of experiment 1

## Experiment 2

The hyperparameter of Pooling layers is given the values of [25500, 12500, 6500]. This is the only difference between this and the previous experiment. The aim of this experiment is to find which range of pooling layers performs the task better. The experiment was conducted quite a long time. About 30 epochs are performed with around 5000 training steps. When we see the training curve it is obvious the training loss range of this experiment quite low than the previous one. But even after training for this many training steps the loss decrease is very slow. The noisy training curve is very apparent. The test accuracy reaches a maximum of 70% around the final epochs of the training. By analyzing the test samples and its ground-truth figures in the 7.2. It is clear that the network performed some useful work on the test set. There is a quite distinction between segmentation parts of certain samples. This shows that the network is learning the segmentation task to some extent and changes in the hyperparameter values may improve the learning of the network.

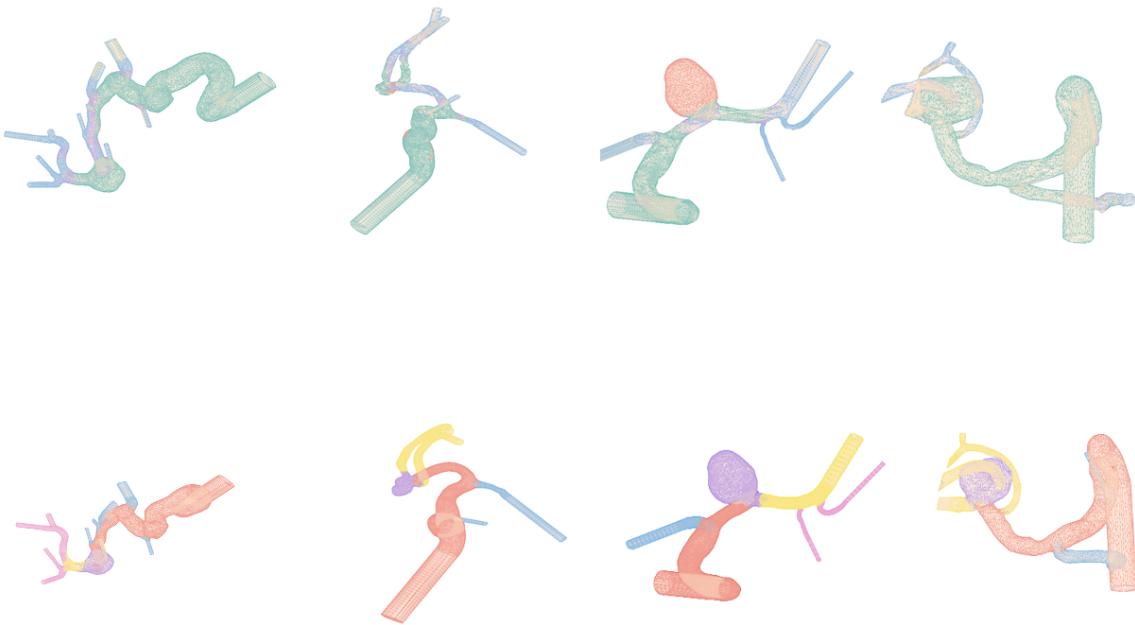


Figure 7.2.: Comparison of test samples (top row) and ground-truth (bottom row) of experiment 2

### Experiment 3

It was clear that the pooling layers can be made the range of values from 24500,12500 to 6500. The network has the capacity to learn for these values. So the values of the hyperparameters of learning rate and residual blocks changes and the experiment are performed on the network. The experiment was conducted for many epochs and the training loss and test accuracy curves are visualized. From the learning curves, it is clear that network learning is far better than any of the previous experiments. It has to be noted that resblock is made to the value 4 which increased the parameters of the network into 2-folds more than the previous experiments and also learning rate was reduced half of the previous experiments. The whole training loss values and test accuracy are added in the appendix.

The output of the test samples shows the learned network does the better segmentation performance and learned to segment all the segmentation parts to an extent. From figure 7.3it is very clear that the network learned to do segmentation of segments 1,2 and 3 significantly. There are better segmentation samples in the figures with a clear distinction between its segmented parts.

From this experiment, we can conclude that the MeshCNN with a bigger architecture can perform well on the segmentation task. The low learning rate also has influence but further investigation is needed to make the conclusion.

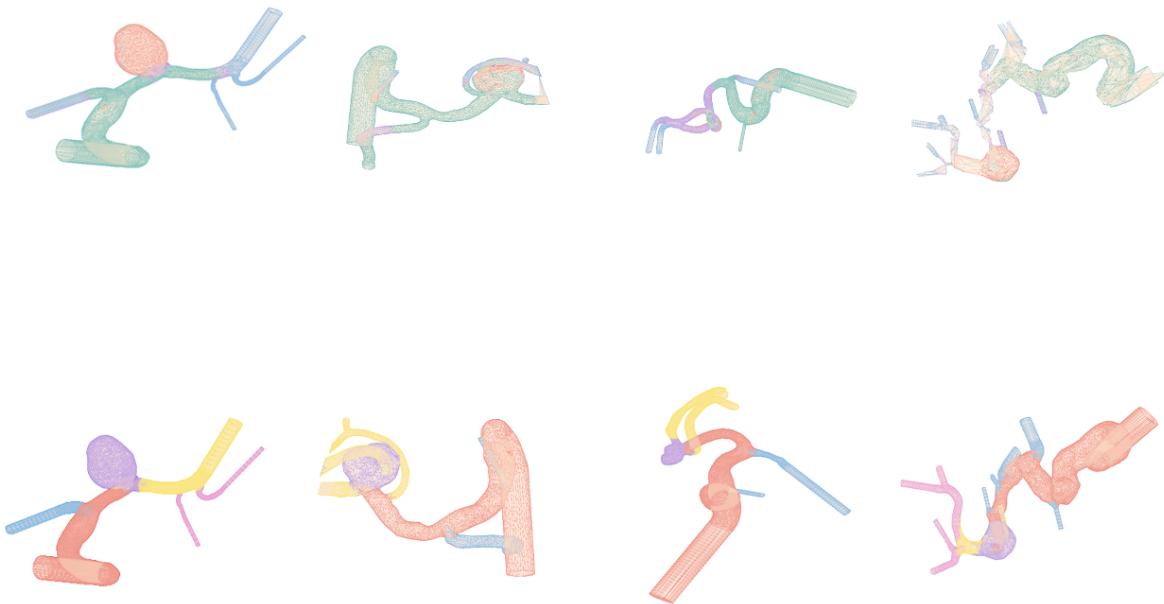


Figure 7.3.: Comparison of test samples (top row) and ground-truth (bottom row) of experiment 3)

#### **Experiment 4**

This experiment was conducted to check how the augmentation technique called vertice scaling affects the learning of the network. The inclusion of vertice scaling and no vertice sliding is the only difference in the hyperparameter values of this experiment with the default hyperparameter list.

The experiment was carried out to a similar period to experiment 3. But the network size is small because of the use of only 3 residual blocks.

From the training and test curve it is apparent that the learning was poor compared to any other experiment. This shows that the vertice scaling concept has no big influence on the learning improvement of the network.

The test accuracy didn't reach more than 55 %. The test samples are shown in figure 7.4 didn't produce any significant learning. But just the predicts the most part of aneurysm to segment 1 and few samples also predict the segments 3 part as the different segment. But there is nothing much impressive about it.

#### **Experiment 5**

This is the last experiment conducted to finally check the influence of the vertice scaling concept on the network performance. The only change in the hyperparameter from the past experiment is that pooling layer range. The pooling was not done ambitiously to a great range just to check if that may affect the performance in any way with the vertice scaling technique.

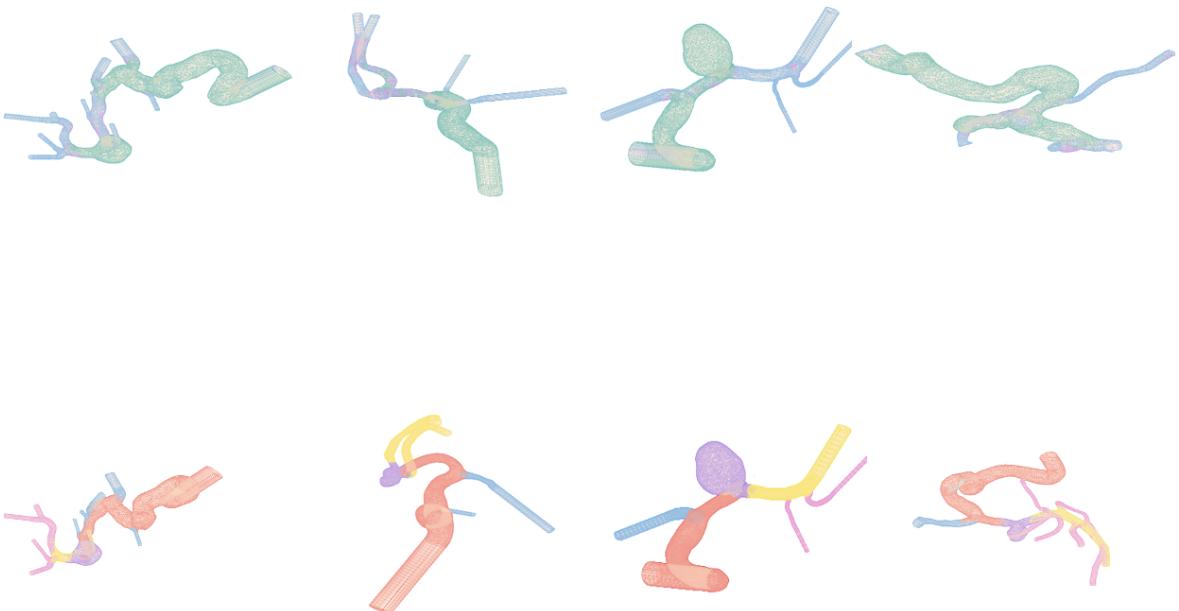


Figure 7.4.: Comparison of test samples (top row) and ground-truth (bottom row) of experiment 4

But by seeing the learning curves and the test samples, it is obvious that there is no significant improvement in the learning of the network.



Figure 7.5.: Comparison of test samples (top row) and ground-truth (bottom row) of experiment 5

## **Summary**

From the analysis of all the experimental results, there are many things to be considered. Firstly it is clear that MeshCNN architecture is robust enough to learn a complex segmentation task of aneurysm dataset. But to narrow out the perfect hyperparameter values for the training of this task. It is needed to conduct many more experiments with different hyperparameter values.

Though there is significant progress in the learning of the network it is apparent that there is a long way for the network to reach the high training and test accuracy performance. It is clear that there is an underfitting in the learned network. This implies that the network needs a lot more training samples.

Finally, we can conclude that the MeshCNN is a robust and efficient network for the aneurysm segmentation task and its performance and accuracy can be improved with more training samples, better augmentation techniques and less training time.

## 8. Conclusion

### 8.1. Summary

All the goals in the project description are implemented. Firstly, a comprehensive literature review is made to understand the various available classical mesh segmentation techniques and then different types of 3D non-euclidean data and its applications are explored.

Many 3D deep learning publications related to segmentation tasks are taken into account and some of the popular and well-performed literature that works better on 3D data is added with its advantages and limitations.

A preliminary experiment was conducted with the human segmentation dataset with the pre-trained MeshCNN to understand the complexity of MeshCNN with the dataset of samples with varied input edges. The output of the dataset showed that the MeshCNN is sophisticated enough to work with various edge size. Then, the next steps were performed to generate the ground-truth data for the aneurysm dataset. The labeling concept was generated for the segmentation task of aneurysm dataset.

As there is an easy way to generate the new edge-based ground-truth for an aneurysm. It was needed to do exhaustive manual labeling of segmentation parts on each of the aneurysm samples. The label generation of the segments is performed in blender 2.8 software with the python interface.

Then, the correctness of the ground-truth data files is checked and the results are visualized and assured.

The major limitation of the medical dataset was less availability of data for deep learning applications. The popular method to alleviate this problem is to incorporate augmentation techniques to maximize the training samples. Various augmentation techniques that are currently available for mesh data were reviewed. It is clear that there is very less number of efficient augmentation methods are available for 3D-Data like meshes. Hence, a simple manual augmentation technique was performed to generate augmented samples for the training of aneurysm.

Finally, Various experiments are conducted on the ground-truth generated aneurysm dataset and the performance of each of the experiment are analyzed. From the results of the experiment, it is obvious that MeshCNN is robust enough to learn the large-sized and complex structures like an aneurysm. The training curves show an incremental decrease in training loss. And the output of some experiments also shows the better learning of the segment parts of the aneurysm.

But, It can be seen from the learning curve that there is room for improvement in the learning of the task by the network.

Some of the reasons for this limitation are the limited number of training samples for this large size of the data sample, less robust augmentation technique and also longer training time which limits the search of the finer hyper-parameter tuning.

Some ideas and suggestions are proposed in the next section that can potentially improve the learning of the network for the segmentation task of aneurysm dataset.

## **8.2. Future and Recommendations**

The experimental results showed impressive results for the aneurysm dataset. This makes it clear that the MeshCNN is robust to understand the proposed segmentation task of the complex structured aneurysm dataset. The training loss and test accuracy of the experiments are noisy but still, the smoothed visualization of the experimental results shows the network is sophisticated enough to perform the learning of the task in increments.

The next step can be to analyze the robustness of the network for the aneurysm segmentation problem. The proposed segmentation parts consist of any number of segmentation parts that make the learning process of the network heavy. The investigation of segmentation results for the simpler segmentation concept will show some lights about which segmentation parts are harder for the network to learn.

Another limitation that is seen during work is the size of the aneurysm sample. There is a trade-off between the size of the aneurysm mesh and the loss of structural information. The complex structure like aneurysm needs a large number of edges to provide smooth and accurate geometric information.

It would be better to integrate the network with powerful parallel computing techniques that may reduce the training times to some folds.

### **More Data**

The major limitation for the training of the network is the less number of available ground-truth samples for the segmentation task. This problem can be solved by implementing new augmentation techniques. But currently, the number of augmentation techniques on non-euclidean data is far less than euclidean data. New augmentation techniques in the near future will increase the generalization of the network.

Transfer learning can alleviate the problem of slow training and lack of training data to a large extent. Transfer learning is the concept of transferring the whole or certain layers of the pre-trained network for our task.

Many experiments have shown that the pre-trained network boost up the training to a big extent. The next steps can be performed aneurysm segmentation task on the complete or partially pre-trained network and the results can be analyzed.

### **Data generation techniques**

Generative Adversarial Networks (GANs) are known for generating adversarial samples. There are few papers that deal with the generation of adversarial meshes. These concepts are quite tricky because to generate new samples the GANS that works on mesh must produce the adversarial samples for the less number of training data. There are interesting mesh generating adversarial networks such are CoMA [36], MeshGAN [37] and MeshAdv [38].

.

### **Adversarial Robustness**

Another interesting terrain would be improving the stability of the neural network for the adversarial perturbation. The network can be made to expose to the adversarial mesh attacks and train the network to learn the adversarial errors and learn to improve its robustness. [39] is one of the few mesh attack and defense publication that was proposed for non-euclidean data.

To sum up, the next steps in the near future can be the development of new complex augmentation techniques that would increase the generalization of the network, Analysis of the meshCNN with transfer learning to see how well the network performance can be boosted with the pre-trained weights. Finally, incorporation of the sophisticated adversarial concepts to improve the generalization of network and improvement of accuracy again adversarial perturbation. And development and usage of new mesh data generation algorithm that can solve the limited availability of training sample problem.

## Bibliography

- [1] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: A network with an edge, 2018.
- [2] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.
- [3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
- [4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.
- [5] Ariel Shamir. A survey on mesh segmentation techniques. *Computer Graphics Forum*, 27(6):1539–1556, 2008.
- [6] Eman Ahmed, Alexandre Saint, Abd El Rahman Shabayek, Kseniya Cherenkova, Rig Das, Gleb Gusev, Djamilia Aouada, and Bjorn Ottersten. A survey on deep learning advances on different 3d data representations, 2018.
- [7] Dina Khattab, Hala M. Ebeid, Ashraf S. Hussein, and Mohamed F. Tolba. 3d mesh segmentation based on unsupervised clustering. In Aboul Ella Hassanien, Khaled Shaalan, Tarek Gaber, Ahmad Taher Azar, and M. F. Tolba, editors, *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2016*, pages 598–607, Cham, 2017. Springer International Publishing.
- [8] Marcelo Santos and Luciano Oliveira. Isec: Iterative over-segmentation via edge clustering, 2018.
- [9] Zongji Wang and Feng Lu. Voxsegnet: Volumetric cnns for semantic part segmentation of 3d shapes, 2018.
- [10] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation, 2016.
- [11] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H. S. Torr. Conditional random fields as recurrent neural networks, 2015.

- [12] Robin Brügger, Christian F. Baumgartner, and Ender Konukoglu. A partially reversible u-net for memory-efficient volumetric image segmentation, 2019.
- [13] Toan Duc Bui, Jitae Shin, and Taesup Moon. 3d densely convolutional networks for volumetric segmentation, 2017.
- [14] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition, 2015.
- [15] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2016.
- [16] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017.
- [17] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds, 2018.
- [18] Huan Lei, Naveed Akhtar, and Ajmal Mian. Spherical kernel for efficient graph convolution on 3d point clouds, 2019.
- [19] Alexandre Boulch. Convpoint: Continuous convolutions for point cloud processing, 2019.
- [20] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters, 2018.
- [21] Eric-Tuan Le, Iasonas Kokkinos, and Niloy J. Mitra. Going deeper with point networks, 2019.
- [22] Peng Jiang and Srikanth Saripalli. Lidarnet: A boundary-aware domain adaptation model for lidar point cloud semantic segmentation, 2020.
- [23] Oren Rippel, Jasper Snoek, and Ryan P. Adams. Spectral representations for convolutional neural networks, 2015.
- [24] Li Yi, Hao Su, Xingwen Guo, and Leonidas Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation, 2016.
- [25] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds, 2018.
- [26] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns?, 2019.

- [27] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs, 2017.
- [28] Zhidong Liang, Ming Yang, and Chunxiang Wang. 3d graph embedding learning with a structure-aware loss function for point cloud semantic instance segmentation, 2019.
- [29] Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. Dilated point convolutions: On the receptive field size of point convolutions on 3d point clouds, 2019.
- [30] Jonathan Masci, Davide Boscaini, Michael M. Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds, 2015.
- [31] Adrien Poulenard and Maks Ovsjanikov. Multi-directional geodesic neural networks via equivariant convolution, 2018.
- [32] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels, 2017.
- [33] Yi-Ling Qiao, Lin Gao, Jie Yang, Paul L. Rosin, Yu-Kun Lai, and Xilin Chen. Laplacianet: Learning on 3d meshes with laplacian encoding and pooling, 2019.
- [34] Hao Pan, Shilin Liu, Yang Liu, and Xin Tong. Convolutional neural networks on 3d surfaces using parallel frames, 2018.
- [35] P. Hübner, M. Weinmann, and S. Wursthorn. Voxel-based indoor reconstruction from hololens triangle meshes, 2020.
- [36] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J. Black. Generating 3d faces using convolutional mesh autoencoders, 2018.
- [37] Shiyang Cheng, Michael Bronstein, Yuxiang Zhou, Irene Kotsia, Maja Pantic, and Stefanos Zafeiriou. Meshgan: Non-linear 3d morphable models of faces, 2019.
- [38] Chaowei Xiao, Dawei Yang, Bo Li, Jia Deng, and Mingyan Liu. Meshadv: Adversarial meshes for visual recognition, 2018.
- [39] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense, 2019.

## A. Appendix: Training data of best experiment

### A.1. Experiment 3: Training loss

```
===== Training Loss (Fri Jan 31 03:07:10 2020) =====
===== Training Loss (Fri Jan 31 03:07:24 2020) =====

(epoch: 1, iters: 170, time: 71.877, data: 0.000) loss: 1.138

(epoch: 2, iters: 174, time: 71.562, data: 0.422) loss: 1.299

(epoch: 3, iters: 168, time: 71.937, data: 0.078) loss: 1.611

(epoch: 4, iters: 172, time: 72.385, data: 0.617) loss: 1.505

(epoch: 5, iters: 176, time: 34.994, data: 0.078) loss: 1.358

(epoch: 6, iters: 170, time: 64.653, data: 0.000) loss: 1.315

(epoch: 7, iters: 174, time: 72.429, data: 0.565) loss: 1.399

(epoch: 8, iters: 168, time: 71.306, data: 0.094) loss: 1.223

(epoch: 9, iters: 172, time: 71.928, data: 0.000) loss: 1.535

(epoch: 10, iters: 176, time: 35.090, data: 0.085) loss: 1.089

(epoch: 11, iters: 170, time: 71.932, data: 0.078) loss: 1.249

(epoch: 12, iters: 174, time: 70.767, data: 0.000) loss: 1.169

(epoch: 13, iters: 168, time: 71.254, data: 0.078) loss: 1.302

(epoch: 14, iters: 172, time: 72.142, data: 0.000) loss: 1.417
```

(epoch: 15, iters: 176, time: 34.873, data: 0.078) loss: 1.040

(epoch: 16, iters: 170, time: 74.846, data: 0.000) loss: 1.079

(epoch: 17, iters: 174, time: 71.610, data: 0.551) loss: 1.192

(epoch: 18, iters: 168, time: 72.026, data: 0.078) loss: 1.332

(epoch: 19, iters: 172, time: 71.345, data: 0.533) loss: 1.077

(epoch: 20, iters: 176, time: 35.059, data: 0.063) loss: 1.111

(epoch: 21, iters: 170, time: 72.149, data: 0.078) loss: 1.151

(epoch: 22, iters: 174, time: 71.385, data: 0.556) loss: 1.116

(epoch: 23, iters: 168, time: 71.812, data: 0.000) loss: 1.087

(epoch: 24, iters: 172, time: 72.086, data: 0.078) loss: 1.343

(epoch: 25, iters: 176, time: 35.248, data: 0.486) loss: 1.277

(epoch: 26, iters: 170, time: 71.704, data: 0.078) loss: 1.331

(epoch: 27, iters: 174, time: 71.789, data: 0.078) loss: 1.116

(epoch: 28, iters: 168, time: 71.977, data: 0.328) loss: 1.023

(epoch: 29, iters: 172, time: 64.846, data: 0.078) loss: 1.062

(epoch: 30, iters: 176, time: 35.174, data: 0.078) loss: 0.970

## A.2. Experiment 3: Test Accuracy

===== Testing Acc (Fri Jan 31 06:32:26 2020) =====  
epoch: 1, TEST ACC: [60.945 %]

===== Testing Acc (Fri Jan 31 10:00:15 2020) =====  
epoch: 2, TEST ACC: [59.272 %]

```
===== Testing Acc (Fri Jan 31 13:28:31 2020) =====
epoch: 3, TEST ACC: [61.442 %]

===== Testing Acc (Fri Jan 31 16:56:44 2020) =====
epoch: 4, TEST ACC: [64.031 %]

===== Testing Acc (Fri Jan 31 20:27:56 2020) =====
epoch: 5, TEST ACC: [62.423 %]

===== Testing Acc (Sat Feb 1 00:01:31 2020) =====
epoch: 6, TEST ACC: [60.608 %]

===== Testing Acc (Sat Feb 1 03:31:25 2020) =====
epoch: 7, TEST ACC: [63.006 %]

===== Testing Acc (Sat Feb 1 07:01:46 2020) =====
epoch: 8, TEST ACC: [63.321 %]

===== Testing Acc (Sat Feb 1 10:32:34 2020) =====
epoch: 9, TEST ACC: [65.156 %]

===== Testing Acc (Sat Feb 1 14:03:48 2020) =====
epoch: 10, TEST ACC: [64.12 %]

===== Testing Acc (Sat Feb 1 17:33:34 2020) =====
epoch: 11, TEST ACC: [62.556 %]

===== Testing Acc (Sat Feb 1 21:05:40 2020) =====
epoch: 12, TEST ACC: [64.944 %]

===== Testing Acc (Sun Feb 2 00:35:48 2020) =====
epoch: 13, TEST ACC: [63.081 %]

===== Testing Acc (Sun Feb 2 04:06:07 2020) =====
epoch: 14, TEST ACC: [63.104 %]

===== Testing Acc (Sun Feb 2 07:35:36 2020) =====
epoch: 15, TEST ACC: [64.078 %]
```

---

A. Appendix: Training data of best experiment

---

```
===== Testing Acc (Sun Feb  2 11:04:20 2020) =====
epoch: 16, TEST ACC: [61.305 %]

===== Testing Acc (Sun Feb  2 14:34:49 2020) =====
epoch: 17, TEST ACC: [62.007 %]

===== Testing Acc (Sun Feb  2 18:04:03 2020) =====
epoch: 18, TEST ACC: [64.299 %]

===== Testing Acc (Sun Feb  2 21:30:14 2020) =====
epoch: 19, TEST ACC: [64.394 %]

===== Testing Acc (Mon Feb  3 00:58:53 2020) =====
epoch: 20, TEST ACC: [59.759 %]

===== Testing Acc (Mon Feb  3 04:27:05 2020) =====
epoch: 21, TEST ACC: [60.468 %]

===== Testing Acc (Mon Feb  3 07:55:16 2020) =====
epoch: 22, TEST ACC: [58.687 %]

===== Testing Acc (Mon Feb  3 11:26:39 2020) =====
epoch: 23, TEST ACC: [66.026 %]

===== Testing Acc (Mon Feb  3 14:59:32 2020) =====
epoch: 24, TEST ACC: [63.308 %]

===== Testing Acc (Mon Feb  3 18:34:12 2020) =====
epoch: 25, TEST ACC: [63.949 %]

===== Testing Acc (Mon Feb  3 22:02:42 2020) =====
epoch: 26, TEST ACC: [66.167 %]

===== Testing Acc (Tue Feb  4 01:32:35 2020) =====
epoch: 27, TEST ACC: [65.989 %]

===== Testing Acc (Tue Feb  4 05:03:43 2020) =====
epoch: 28, TEST ACC: [65.557 %]
```

```
===== Testing Acc (Tue Feb 4 08:30:23 2020) =====
epoch: 29, TEST ACC: [63.889 %]
```

```
===== Testing Acc (Tue Feb 4 12:01:00 2020) =====
epoch: 30, TEST ACC: [67.873 %]
```

### A.3. Experiment 3: Parameters

```
----- Options -----
arch: meshunet
batch_size: 2
beta1: 0.9
checkpoints_dir: ./checkpoints
continue_train: False
dataroot: datasets/aneurysm
dataset_mode: segmentation
epoch_count: 1
export_folder:
fc_n: 100
flip_edges: 0
gpu_ids: []
init_gain: 0.02
init_type: normal
is_train: True
lr: 0.0005
lr_decay_iters: 50
lr_policy: lambda
name: aneurysm2
ncf: [32, 64, 128, 256]
ninput_edges: 30100
niter: 30
niter_decay: 2000
no_vis: False
norm: batch
num_aug: 20
num_groups: 16
num_threads: 3
```

```
phase: train
pool_res: [24500, 12500, 6500]
print_freq: 10
resblocks: 4
run_test_freq: 1
save_epoch_freq: 1
save_latest_freq: 250
scale_verts: False
seed: None
serial_batches: False
slide_verts: 0.1
verbose_plot: False
which_epoch: latest
----- End -----
```