

MASTERARBEIT


3D MESH SEGMENTATION USING TRANSFORMER BASED GRAPH OPERATIONS

Freigabe:

Der Bearbeiter:

Unterschriften

Harinandan Teja Katam



Betreuer:

Maximilian Denninger



Der Institutsdirektor

Prof. Alin Albu-Schäffer



Dieser Bericht enthält 107 Seiten, 40 Abbildungen und 13 Tabellen



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

3D Mesh Segmentation Using Transformer Based Graph Operations

Katam Harinandan Teja





DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

3D Mesh Segmentation Using Transformer Based Graph Operations

3D Mesh Segmentierung Mittels Transformer Basierten Graph Operationen

Author:	Katam Harinandan Teja
Supervisor:	Dr. habil. Rudolph Triebel
Advisor:	Denninger Maximilian
Submission Date:	9th January 2021



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 9th January 2021

Katam Harinandan Teja

Acknowledgments

Firstly I would like to thank the DLR and Dr. Rudolph Triebel, for giving me the opportunity to work on my Master Thesis and expand my practical experience in Deep Learning and Computer Vision. I would also like to thank my supervisor Maximillian Denninger, who has been very supportive throughout my time at DLR. Additionally, I would like to thank the rest of my team at DLR for helping me write my thesis. Last but not least I would like to thank my parents, sister, and brother in law who have been my pillars for every step in my life.

Abstract

Deep Learning and especially convolutions have been a massive success in computer vision tasks such as Segmentation, Object Detection, and others. However, all of these are limited to 2D images, whereas the progress in the 3D domain has been limited. Extending these priors works to the 3D domain is sadly not straightforward. The biggest challenge here is the unstructured representation of 3D data such as meshes or point clouds. While other works use voxel grids, which have structured representation, they usually struggle with computation time and memory.

In this thesis, the task of extending a convolution operation to unstructured data and its problems along with potential solutions is explored. In this thesis, two new methods to the task of Mesh Segmentation are proposed. Both these methods are based on Transformer networks and their components. In the first method, a first of its kind application of transformers to the task of Mesh Segmentation is proposed. In the second method, a permutation invariant Graph Convolution layer named Transformer Convolution (TransConv) is proposed which acts similar to a convolution operation on images and can be used in any model architecture. In addition to these methods, two extensions are proposed that improve the performance of both our methods. The first extension is to use depth encoding to add more information about the geodesic distance to the model. The second extension is to extend the concept of atrous convolutions in images to meshes.

All of our methods and extensions are evaluated on two datasets and compared with other related works. The first dataset is a collection of high-resolution meshes called the Shape COSEG (COSEG) dataset. The second dataset is a collection of point clouds of 3D objects called ShapeNet part annotation.

Our proposed graph convolution layer TransConv outperforms other related works in both the datasets. However, our method to use transformers for mesh segmentation produced comparable results.

Abstract - German

Deep Learning und insbesondere Convolutions hatten große Erfolg bei typischen Computer Vision Aufgaben wie z. B. Segmentierung, Objekterkennung und anderen. Allerdings sind alle diese Ansätze auf 2D-Bilder beschränkt, während hingegen die Fortschritte im 3D-Bereich begrenzt sind. Das Erweitern dieser Arbeiten auf die 3D-Domäne, ist leider nicht ohne Probleme möglich. Die größte Herausforderung dabei ist die unstrukturierte Darstellung von 3D-Daten, wie z. B. in Mesh oder Punktwolken. Während andere Arbeiten Voxel verwenden, welche eine strukturierte Repräsentation haben, haben sie jedoch in der Regel Probleme mit der Rechenzeit und Speicherplatznutzung.

In dieser Arbeit werden Convolutions auf unstrukturierte Daten erweitern und deren möglichen Probleme und darauffolgenden Lösungen erforscht. Hierbei werden zwei neue Methoden für die Aufgabe der Mesh-Segmentierung vorgeschlagen. Beiden Methoden basieren dabei auf Transformer Netzwerken und deren Komponenten. In der ersten Methode wird zum ersten Mal eine Anwendung von Transformatoren für die Aufgabe der Mesh-Segmentierung vorgestellt. In der zweiten Methode wird eine permutationsinvariante Graphenfaltungsschicht namens Transformer Convolution vorgeschlagen, die sich ähnlich wie eine Convolution auf Bildern verhält und in jeder Modellarchitektur verwendet werden kann. Zusätzlich zu diesen Methoden werden zwei Erweiterungen vorgeschlagen, die die Leistung unserer beiden Methoden verbessern. Die erste Erweiterung ist die Verwendung von einer Tiefenkodierung, um mehr Informationen über die geodätische Distanz zum Modell hinzuzufügen. Die zweite Erweiterung ist die Erweiterung einer Atrous Convolution von Bildern auf Graphen. Alle unsere Methoden und Erweiterungen werden mit zwei Datensätzen evaluiert und mit anderen verwandten Arbeiten verglichen. Der erste Datensatz ist eine Sammlung von hochauflösenden Meshes, der Shape COSEG-Datensatz. Der zweite Datensatz ist eine Sammlung von Punktwolken von 3D-Objekten genannt ShapeNet.

Die von uns vorgeschlagene Graphen Convolution Schicht TransConv übertrifft andere verwandte Arbeiten in beiden Datensätzen. Unsere Methode zur Verwendung von Transformatoren für die Meshsegmentierung lieferte zudem vergleichbare Ergebnisse.

Contents

Acknowledgments	iii
Abstract	iv
Abstract - German	v
1 Introduction	1
1.1 Problem Statement	4
1.2 Thesis Structure	5
2 Related Work	6
2.1 Classical Computer Vision Methods	6
2.2 Projective analysis	7
2.3 Graph analysis	9
2.3.1 Recurrent Graph Neural Networks (RecGNNs)	9
2.3.2 Convolutional Graph Neural Networks (ConvGNNs)	10
2.3.3 Graph Autoencoders (GAEs)	12
3 Spatial-based Graph Convolutions	13
3.1 Ordered to Unordered Data	14
3.1.1 Permutation Invariance	14
3.2 Feature-Steered Graph Convolutions (FeaStNet)	15
3.3 MeshCNN	15
3.4 Graph Attention Network (GAT)	16
4 Transformers	20
4.1 Sequence-to-Sequence (Seq2Seq)	20
4.2 Attention	21
4.2.1 Generalized Attention	22

4.2.2	Alignment Model	23
4.3	Transformer Network	24
4.3.1	Encoder	26
4.3.2	Decoder	26
4.3.3	Positional Encoding	28
4.3.4	Scaled Dot Product Attention	30
4.3.5	Multi-Head Attention	31
4.3.6	Self Attention	32
4.3.7	Position-wise Feed-Forward Networks	33
4.4	Examples	33
4.4.1	Machine Translation	34
5	Our Approach	37
5.1	Problem Statement Recap	37
5.2	Transformer for Mesh Segmentation (MeshTrans)	37
5.3	Transformer Convolution (TransConv)	39
5.3.1	Set Transformer	42
5.3.2	Final Architecture	45
5.3.3	Support for Batching	45
5.4	Depth Encoding	45
5.5	Atrous Convolution	48
5.5.1	Depth Encoding	50
6	Experimental Setup	52
6.1	Environment	52
6.2	Datasets	52
6.2.1	COSEG	53
6.2.2	ShapeNet	56
6.3	Models	58
6.3.1	Transformer	59
6.3.2	FeaStNet	62
6.3.3	U-Net	63
6.4	Training	66
6.4.1	Loss	66

6.5	Evaluation	66
6.6	Implementation	68
6.6.1	Batching	68
6.6.2	Breadth First Search	68
7	Results	69
7.1	Model configurations	69
7.2	The Shape COSEG	70
7.3	ShapeNet	72
7.3.1	Depth Encoding	73
8	Future Steps	79
8.1	Depth Encoding strategies	79
8.2	Datasets	79
8.3	Memory Consumption	80
9	Conclusion	81
	List of Figures	82
	List of Tables	87
	Bibliography	89
	List of Abbreviations	95

1 Introduction

In the past few years, there has been a rising interest and demand for 3D data modeling. One clear instance of the rise is in the construction field where people are moving away from paper plans to Building Information Modeling (BIM) to generate construction plans. The UK government mandated BIM in April 2016 in every construction project which requires that all projects funded by the central government be delivered with ‘fully collaborative 3D BIM’. As the mandate has come into force, there has been a rise in levels of BIM adoption. According to the National BIM Report 2018, 20% of the industry has adopted BIM since 2016 mandate. Also, many mechanical parts used in the industrial equipment are now being designed using various 3D modeling software instead of 2D sketches.

Autonomous systems such as autonomous driving, autonomous robots moved from being science fiction to a very real possibility during the past twenty years. Any such autonomous system that interacts with the real world need a 3D model of the environment to make decisions and perform tasks. There are many other fields where more and more 3D modeling is being adapted.

With the increase in usage of 3D data, there has also been an increase in the need for understanding of the 3D data. A simple example could be that you have a 3D model of a mechanical part and want a system to automatically segment the part into multiple components as shown in figure 1.1. Another instance where interpreting 3D data is required is in autonomous driving where the system should have a semantic understanding (as shown in figure 1.2) of its environment to make decisions.

How a 3D model is represented plays an important role in developing techniques that can understand the model. There are many different representations in which a 3D model can be represented. Following are the three most used formats in which 3D information is represented.

- **Voxel grids** : Voxel grids is a geometry type defined on a regular 3D grid similar to images in 2D. A voxel can be thought of as the 3D counterpart to a pixel in 2D.

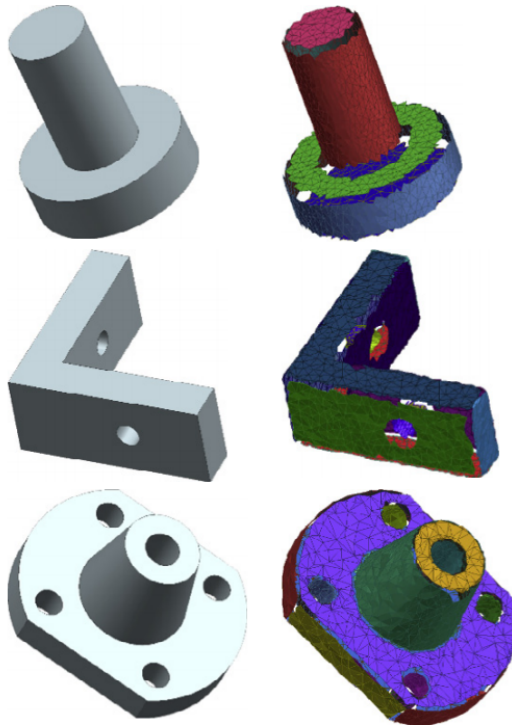


Figure 1.1: Shown an example of segmentation of a mechanical part. The image is from [Buo+17].

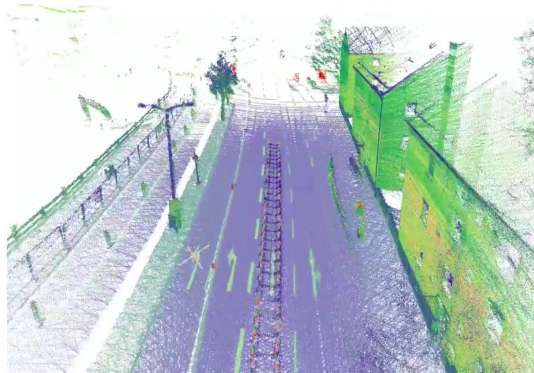


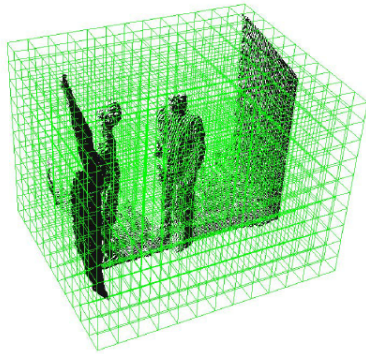
Figure 1.2: Shown an example of 3D understanding of your environment during autonomous driving. The image is from [Mit19].

Although they closely resemble images in structure, they are not widely used due to the memory requirement to store the voxel grid. In most voxel grids the subject takes only up to 50% of the grid and the rest of the grid is usually empty hence requiring

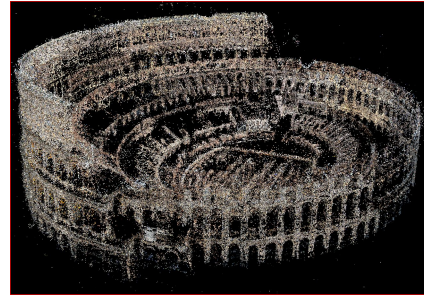
more memory than necessary. Furthermore, the resolution is limited.

- **Meshes** : A mesh is a collection of vertices, edges and faces that defines the shape of an object. Meshes can easily convey the distinct identities of a mesh through geodesic separation, despite their proximity in euclidean space. Figure 1.4 shows one such example on a camel mesh.
- **Point Clouds** : Point cloud is a set of data points in space. The points represent a 3D shape or object. Point clouds are the most popular and also the simplest of all the representations for 3D. Nowadays, point clouds can be easily obtained using LiDAR or Depth sensors or various other sensors. But unlike meshes point clouds do not have any information about the shape structure of the subject due to no information about how the points are connected.

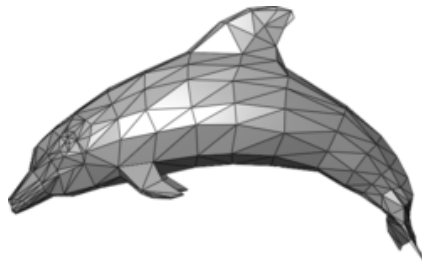
Figure 1.3 shows examples for each of the three data structure types.



(a) Voxel Grid.



(b) Point Cloud.



(c) Mesh.

Figure 1.3: Different data structures for 3D data. The images are from [WLX12], [Aga+11] and [Pol10].

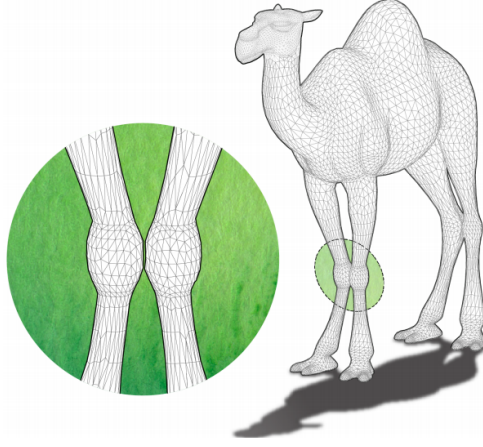


Figure 1.4: An example where meshes can easily convey the distinct identities of the camel joints through geodesic separation, despite their proximity in euclidean space. The image is from [Han+19].

While there are many works [CMZ19], [DT20] that use voxel grids for 3D understanding, they usually struggle with computation time and memory. Due to the ease in obtaining and simplicity in its representation most of the recent effort is focused on developing techniques [Qi+17a], [Qi+17b], [Li+18] for point cloud shape analysis. Due to the advantages that meshes have over other data structures, meshes are used as the data structure for 3D models in our thesis.

1.1 Problem Statement

Over the past decade, there has been a massive success of deep learning in image understanding tasks such as Segmentation [Che+17], Object Detection [Ren+16], [TPL20], Instance Segmentation [He+18] and others. However, all of the success has been limited to 2D images, whereas the progress in the 3D domain has been limited. Most of the algorithms in images rely on the Convolutional Neural Networks (CNNs) and extending these works to 3D data is not straightforward. The biggest challenge here is the unstructured representation of 3D data such as meshes or point clouds.

Hence in this thesis, the task of extending convolution operations to unstructured data and their application to the task of Mesh segmentation are explored. Mesh segmentation can be

formally defined as for any given Mesh or Graph $G : (X, E)$, where $X : \{x_1, \dots, x_N\}$, $x_i \in \mathbb{R}^F$ are the features of the vertices and $E \in [0, \dots, N]^2$ are the edges between the vertices, assign each vertex i a label y_i from a predefined set of categories of length C .

1.2 Thesis Structure

This thesis is structured as follows. In chapter 1, an introduction to 3D data modeling and different 3D representations is provided along with the problem statement of this thesis and the thesis structure. In chapter 2 an overview of existing approaches to the problem statement is given, highlighting the differences to our approach. Next in chapter 3 existing works that are very closely related to our approaches are discussed in detail. Afterwards, in chapter 4, transformer networks [Vas+17] and its various components are explained. Also at the end of the chapter, an example of applying a transformer network to Machine Translation is explained. In chapter 5, our approaches to the task of Mesh Segmentation are discussed in detail. Afterwards, in chapter 6, the experimental setup for the thesis which includes the datasets used, model architectures used, training and evaluation procedures, and a few implementation details are discussed. In chapter 7, results of our approaches and extension works are listed and interpreted. Later in chapter 8, future steps and ideas that are out of scope for this thesis but are related to our work are discussed. At the end in chapter 9, our work in this thesis is summarized.

2 Related Work

This chapter provides an overview of existing 3D mesh segmentation techniques. First a brief overview of traditional 3D segmentation techniques that calculate handcrafted features and use classical Machine Learning models to form clusters is provided. Followed by some of the image-based 3D mesh techniques where images of the 3D object at interest are used to segment the object in 3D. At the end techniques that directly operate on the 3D data of the model using Deep Learning to generate segmentation and the categorization of such techniques are discussed. Our approach falls into the last category of these techniques as meshes are directly used in our approach to generate segmentations.

Most of the techniques that operate directly on the 3D mesh use some form of Graph Neural Networks. Such techniques use the graph nature of the meshes to transfer information among the vertices/edges of the mesh, then use this information to segment the vertices/edges. A more detailed discussion on Convolutional Graph Neural Network (ConvGNN) is presented in section 3.

2.1 Classical Computer Vision Methods

Classical Computer Vision techniques involve creating hand crafted features and training classifiers using these features that can segment a 3D mesh. In [KHS10] a CRF (Conditional Random Field) model is designed for the task of mesh segmentation. The objective function then consists of two energies, the unary energy and the geometric pair wise energy. The first energy corresponds to the probability of assigning a label to a face given the input features which include descriptors of local surface geometry and context, such as curvatures, shape diameter, and shape context. The second energy term corresponds to whether adjacent faces should have the same label given the input features which include pairwise features such as dihedral angles, which helps improve the accuracy at the boundaries.

Techniques like [KHS10] that use CRF assumes that the labeling of a node depends only on its neighbors and with no message propagation steps between the neighbors the receptive

field is constant. Such assumptions do not work when the mesh resolution is increased. Also, the geometric pairwise energy term assumes that near the boundaries where the labels change there is a significant change in the geometric features, which again might not be the case with every mesh. In our work depending on the depth of the network the receptive field increases exponentially thus allowing the network to gather more information before assigning the labels.

In [SC08] Shape Diameter Function (SDF) is used to generate segmentations. Given a mesh, the SDF provides an estimate of the local object diameter for each facet of the mesh (the SDF values). SDF for a given point is calculate by projecting several rays from inside a cone centered around its inward-normal direction and calculating the weighted average of the lengths of rays that intersect with the other side of the mesh.

The segmentation algorithm first applies a soft clustering on the facets using the associated SDF values. The final segmentation is then obtained via a graph-cut algorithm that considers surface-based features (dihedral-angle and concavity) together with the result of the soft clustering.

In this work, an assumption that meshes are closed is made, because in an open mesh for a few points the rays projected to calculate the SDF values do not intersect. Also, this method fails in the boundary regions as the SDF values do not vary much around the label boundaries. In our approach, no such assumptions about meshes are made.

2.2 Projective analysis

Image-based techniques involve either directly using the 2D images of the 3D model from various datasets or projecting 3D model into 2D images from different viewing angles. These images can be various renderings/captures of the 3D object for the corresponding viewing angle such as the color or depth. These images are passed through Neural Networks to generate features that are then projected to 3D using the camera parameters (intrinsic and extrinsic). These projected features are then passed through some classifiers (Deep Learning or Classical) to calculate the final segmentation.

ShapePFCN (3D Shape Segmentation with Projective Convolutional Networks) [Kal+17] combines image-based Fully Convolutional Network (FCN) and surface-based Conditional Random Field (CRF) to yield segmentations of 3D shapes. It first applies FCN's on color and depth images of the 3D model from different viewpoints to calculate the per-label

confidence maps for all the viewpoints. It then uses an Image2SurfaceProjection layer to project the label maps from different viewpoints onto the 3D model. Finally, it uses a CRF layer to generate 3D labels.

[Wan+13] treats an input 3D shape as a collection of 2D projections. It first labels each projection by transferring knowledge from existing labeled images, and then back-projects and fuses the labelings on the 3D shape. For a given 3D shape they produce a set of multi-view projections as binary images. Each projection is used to retrieve multiple images from semantically labeled images (from ImageNET dataset) based on a novel bi-class Hausdorff distance, label projection by performing label transfer and an associated confidence map. All labeled projections and confidence maps are back-projected onto the input 3D model to compute the labeling probability map. Finally, graph cut segmentation is applied based on the labeling probabilities to produce the final segmentation and labeling.

[Law+17] uses a similar approach of projecting 3D input into 2D images, applying a Neural Network to segment the 2D images which are then fused and back-projected to 3D to form the final segmentation. They demonstrate the application on a scene point cloud.

Projective analysis simplifies the processing task by working in a lower-dimensional space, circumvents the requirement of having complete and well-modeled 3D shapes, and in few cases addresses the data challenge for 3D shape analysis by leveraging the massive available image data. Such techniques are more memory efficient as they operate on lower-dimensional data and usually share the weights of the FCN's applied on the images. But one major problem with image-based techniques is that they need to select viewpoints such that when combined they cover 100% of the 3D model. Not doing so will produce results that are not accurate in the parts which were not visible from any of the viewpoints. Selecting such viewpoints is a problem of its own. One solution is to select a huge number of viewpoints and hope that they cover 100% of the 3D model. But then this increases inference times as there are as many images as the number of viewpoints to process and generate segmentations. Another problem with image-based techniques is the difficulty to perform real-time 3D segmentation as they need images from multiple viewpoints that cover the entire 3D model. In our approach, the algorithm is directly applied on the 3D model hence with sufficient optimizations can be run in real-time.

2.3 Graph analysis

The next category of techniques make use of the underlying graph representation of 3D meshes to extract neighborhood information and perform Deep Learning operations on them. As per the survey on Graph Neural Networks in [Wu+20] this category of techniques are mainly divided into three sub-categories

- Recurrent Graph Neural Networks (RecGNNs)
- Convolutional Graph Neural Networks (ConvGNNs)
- Graph autoencoders (GAEs)

2.3.1 Recurrent Graph Neural Networks (RecGNNs)

Recurrent Graph Neural Networks (RecGNNs) [Dai+18] [GM10] [Li+17] [Sca+09] aim to learn node representations with recurrent neural architectures. They assume a node in a graph constantly exchanges information/message with its neighbors until a stable equilibrium is reached. They apply the same set of parameters recurrently over nodes in a graph to extract high-level node representation. RecGNNs updates nodes' states by exchanging neighborhood information recurrently until a stable equilibrium is reached.

In Graph Neural Network (GNN) [Sca+09], for a node v at time t the state $h_v^{(t)}$ is calculated as follows

$$h_v^{(t)} = \sum_{u \in \mathcal{N}_v} f(x_v, x_{(v,u)}^e, x_u, h_u^{(t-1)}) \quad (2.1)$$

where $x_{(v,u)}^e$ is the edge feature for the edge (v, u) , x_v , x_u are the input features of the nodes v and u , $h_u^{(t-1)}$ is the previous state of the node u , $f(\cdot)$ is a parametric function (eg. a neural network) and \mathcal{N}_v is the neighborhood of v .

Graph echo state Network (GraphESN) [GM10] consists of an encoder and an output layer. The encoder is randomly initialized and requires no training. It implements a contractive state transition function to recurrently update node states until the global graph state reaches convergence. Afterward, the output layer is trained by taking the fixed node states as inputs

Gated Graph Neural Network (GGNN) [Li+17] uses a Gated Recurrent Unit (GRU) [Cho+14] as a recurrent function, and fixing the number of recurrence steps. In GGNN equation 2.1 is transformed as shown below

$$h_v^{(t)} = GRU(h_v^{(t-1)}, \sum_{u \in \mathcal{N}_v} W h_u^{(t-1)}) \quad (2.2)$$

where W is a linear transformation applied on previous state of the neighbor u .

The difference between RecGNN and our approach is that, in RecGNN the node states are updated until an equilibrium is reached, whereas in our approach the network consists of a fixed number of layers with different weights and does not wait for any equilibrium, it instead learns the weights by back-propagating the objective loss.

2.3.2 Convolutional Graph Neural Networks (ConvGNNs)

A significant part of Graph Neural Networks are Convolutional Graph Neural Networks (ConvGNNs) which generalize the operation of convolution from grid data to graph data. ConvGNNs are closely related to Recurrent Graph Neural Networks. Instead of iterating node states (using the same weights) until an equilibrium is reached, ConvGNNs uses a fixed number of layers with different weights to address the cyclic mutual dependencies architecturally. Due to the efficiency and the convenience of ConvGNNs to composite with other neural networks, the popularity of ConvGNNs has been rapidly growing in recent years.

According to the survey on Graph Neural Network (GNN) [Wu+20] ConvGNNs fall into two categories spectral-based and spatial-based. Spectral-based approaches define graph convolutions by introducing filters from the perspective of graph signal processing where the graph convolutional operation is interpreted as removing noises from graph signals. Spatial-based approaches [VBV18] [Han+19] [Mic09] [AT16] [NAK16] [Gil+17] inherit ideas from RecGNNs to define graph convolutions by information propagation. Since GCN [KW17] bridged the gap between spectral-based approaches and spatial-based approaches, spatial-based methods have developed rapidly recently due to its attractive efficiency, flexibility, and generality. Our approach falls into the spatial-based convolutions category, as the spatial representation of 3D meshes are used to compute segmentation.

For the interest of this thesis, only spatial-based convolutions are discussed. In this section, a few examples of spatial-based convolutions are discussed and in chapter 3 spatial-based convolutions are discussed in detail.

Neural Networks for Graphs (NN4Gs)

Neural Networks for Graphs (NN4Gs) is one of the first works towards spatial-based graph convolutions. In NN4G a graph convolution is performed by summing up the neighboring

nodes' information directly. NN4G derives its next layer (k) node states h_v^k by

$$h_v^k = f(W_k^T x_v + \sum_{i=1}^{k-1} \sum_{u \in \mathcal{N}_v} \Theta_k^T h_u^{k-1}) \quad (2.3)$$

where W_k, Θ_k are linear transformations, $f(\cdot)$ is an activation function and \mathcal{N}_v is the neighborhood of v .

In NN4G all the transformed neighboring features are given equal weights in the summation, in our approaches the weights to the neighboring features are calculated using the attention mechanism (discussed in detail in the following chapters).

Message Passing Neural Networkss (MPNNs)

Message Passing Neural Networkss (MPNNs) [Gil+17] outlines a general framework of spatial-based ConvGNNs. It treats graph convolutions as a message passing process in which information can be passed from one node to another along edges directly. MPNN runs K-step message passing iterations to let information propagate further. The message passing function (namely the spatial graph convolution) is defined as

$$h_v^k = U_k(h_v^{k-1}, \sum_{u \in \mathcal{N}_v} M_k(h_v^{k-1}, h_u^{k-1}, x_{vu}^e)) \quad (2.4)$$

where $h_v^0 = x_v$ (x_v is the input features for v), $U_k(\cdot)$ and $M_k(\cdot)$ are learnable parameters for step k .

Graph Sage

As the number of neighbors of a node can vary from one to a thousand or even more, it is inefficient to take the full size of a node's neighborhood. GraphSage [HYL18] adopts sampling to obtain a fixed number of neighbors for each node. It performs graph convolutions similar to a MPNN but only on the sampled neighborhood as shown in equation 2.5

$$h_v^k = \sigma(W^k f_k(h_v^{k-1}, \{h_u^{k-1}, \forall u \in \mathcal{S}_{\mathcal{N}_v}\})) \quad (2.5)$$

where $h_v^0 = x_v$ (x_v is the input features for v), $f_k(\cdot)$ is an aggregation function and $\mathcal{S}_{\mathcal{N}_v}$ is sample of fixed number of neighbors for node v .

Graph convolutions such as Graph Sage or MPNN assume identical contributions of neighboring nodes to the central node. In our approaches, the contributions are calculated using the attention mechanism.

2.3.3 Graph Autoencoders (GAEs)

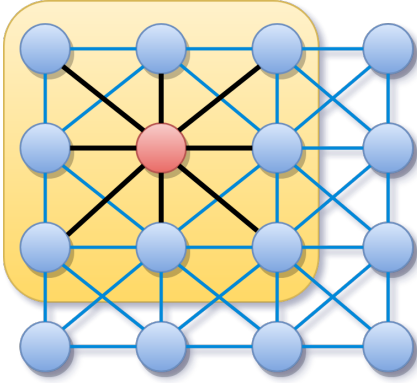
Like auto encoders Graph Autoencoders are unsupervised learning techniques which encode nodes/vertices into latent vector space and reconstruct graph data from the encoded information. Such techniques are used to learn network embeddings and graph generative distributions. These network embeddings can be used to train a classifier to label the nodes thus segmenting the 3D model.

[ZWC19], [WCZ16] [KW16], [Pan+19], [Tu+18] are few of such techniques that propose various types of algorithms to generate Network Embeddings.

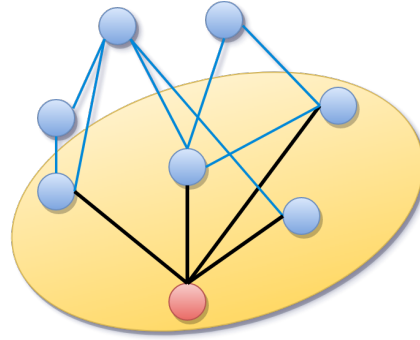
In these techniques, the network embeddings are not specially trained for a specific task such as segmentation. Such embeddings store generalized information about the graphs. In our approaches, the node embeddings are trained for a specific task using the loss for that task, which in our case is the mesh segmentation. Such embeddings store more task-specific information and help produce better results for the task.

3 Spatial-based Graph Convolutions

Spatial-based graph convolutions are analogous to convolution operation of a Convolutional Neural Network (CNN) in an image (2D data). Like in images where the convolutions are based on pixels spatial representation, graph convolutions are based on nodes spatial relation. As shown in 3.1a convolution in images can be considered a special case of a graph convolution where each pixel(node) is connected to its nearby pixels. In 3.1a a filter is applied on a 3x3 patch on the pixel values of the neighboring nodes and the root node. In 3.1b a new root node representation is generated by convolving the existing root node representation with the neighboring node representations.



(a) 3x3 Convolution in 2D.



(b) Graph convolution.

Figure 3.1: Convolution operations on images-2D (left) vs graph-3D (right). Red node is the root node for which convolution is being calculated. Black edges indicate which nodes are being used in the convolution operation. In images when a 3 X 3 filter is applied all the pixel values in the 3 x 3 patch are used to calculate the final value for the root node. In a graph typically all the neighbors of the root node are used to calculate the value for the root node.

3.1 Ordered to Unordered Data

For images, the computation of a convolution operation is as simple as weighted average of the pixel values. The simplicity comes from the structured/ordered data in the images where there is a clear one to one mapping between the weights and the neighbors at relative positions w.r.t the root/center pixel of the convolution (eg. top left to the root node). This is not the same for graphs, as most graphs come without any order i.e. the input to the convolution is not always in the same order like in images and also the number of input nodes vary for each root node. Which makes it difficult to learn weights associated with the nodes.

3.1.1 Permutation Invariance

To overcome this issue most of the graph convolutions are designed to be permutation invariant. Permutation invariance means that for any permutation of an input vector ([A, B, C, D]) like [A, B, D, C], [B, C, A, D] etc. the output of the network should always be the same. In the case of graphs no matter the order of the neighboring nodes the network or the convolutional layer should always return the same output.

To be permutation invariant most of the graph convolutions incorporate at least one of the following into their architecture.

- Permutation invariant operations such as Sum, Max, Min, Mean, etc. where the order of the input does not affect the output
- Decide the order of the input before passing it to the convolution such that the same order is passed in every forward pass
- Train a network on all possible permutations. This is computationally quite expensive and the least preferred option

In our approaches, a series of permutation invariant operations are used to achieve permutation invariance, more details about this is discussed in chapter 5. In the following sections, some of the spatial graph convolution algorithms that are very closely related to our work are discussed in detail and how they manage to be permutation invariant is shown.

3.2 Feature-Steered Graph Convolutions (FeaStNet)

Feature-Steered Graph Convolutions [VBV18] introduces Graph Convolutions using dynamic filters. Instead of assigning a single weight for a pair (i, j) they propose a soft weight assignment $q_m(x_i, x_j)$ as a result the final value of the root node after convolution is defined by

$$x'_i = b + \sum_{m=1}^M \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} q_m(x_i, x_j) W_m x_j \quad (3.1)$$

where x_i, x_j are input features of the nodes i and j respectively, $b \in \mathbb{R}$ is bias, M is the number of weight matrices and $q_m(x_i, x_j)$ is the assignment of x_j to the m -th weight matrix W_m and is calculate as follows

$$q_m(x_i, x_j) \propto \exp(u_m^T x_i + v_m^T x_j + c_m) \quad (3.2)$$

with $\sum_{m=1}^M q_m(x_i, x_j) = 1$. Here W_m, u_m, v_m, c_m are all learnable parameters.

From the equations 3.1 and 3.2 it is clear that the weights are calculated dynamically using the root node and the neighbor representation. Here the permutation invariance is achieved using the summation (\sum) operator. Figure 3.2 shows graphical representation of Feature-Steered Graph Convolution operation where each node in the input patch is associated in a soft manner to each of the M weight matrices based on its features using the weight $q_m(x_i, x_j)$.

Similar to FeaStNet, in our approaches, the weights are calculated dynamically which helps the model to learn the weights associated to their neighbors. However, in our approaches, the weights are calculated not only between the root node and the neighboring nodes, but also among the neighboring nodes themselves. Also, our approaches use a different weight calculation function which is discussed more in detail in chapters 4 and 5.

3.3 MeshCNN

MeshCNN by Hanocka et al. [Han+19] is a convolutional neural network designed specifically for triangular meshes. They use edges as nodes instead of vertices to calculate the 3D mesh segmentation. Convolutions are applied on edges (root) and the four edges of their indicent triangles.

MeshCNN tackles the problem of permutation invariance by aggregating the 1-ring edges into two pairs of edges which have un-ambiguity (e.g. a and c, b and d from 3.3a) and

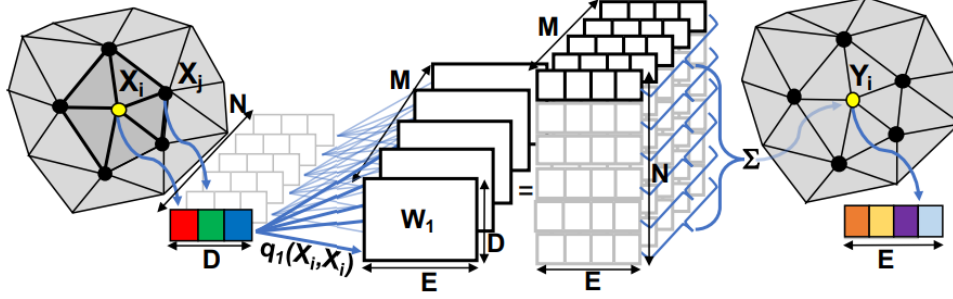


Figure 3.2: Feature Steered graph convolutional network, where each node in the input patch is associated in a soft manner to each of the M weight matrices based on its features using the weight $q_m(x_i, x_j)$. The image is from [VBV18].

generate new features by applying simple symmetric functions on each pair (e.g. $\text{sum}(a, c)$). The convolution is applied on the new symmetric features thereby eliminating any order ambiguity.

The input edge feature for the very first layer is 5-dimensional vector for every edge: the dihedral angle, two inner angles and two edge-length ratios for each face as shown in figure 3.3b. Each of the two face-based features (inner angles and edge-length ratios) are sorted, thereby resolving the ordering ambiguity and guaranteeing invariance.

MeshCNN is an example where the problem of permutation invariance is solved by both using a permutation invariant function and by deciding the order of the input before passing to the convolution.

In MeshCNN edges are used as nodes and the assumption that all the shapes are represented as manifold meshes, possibly with boundary edges is made. Such assumptions are not satisfied when applying the model to point clouds where the edges with the nearest neighbors might not form a polygonal faces. In our work, vertices are used as nodes and no such assumptions about the shapes are made hence making our approaches more extensible.

3.4 Graph Attention Network (GAT)

Graph Attention Network (GAT) by Velickovič et al. [Vel+18] is one of the very first works to use attention mechanism (attention is discussed in detail in 4.2) in a graph convolution

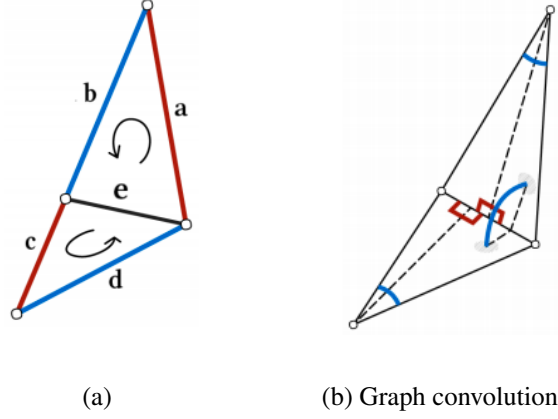


Figure 3.3: (a) The 1-ring neighbors of the edge e can be ordered as (a,b,c,d) or (c,d, a,b) , depending on which face defines the first neighbor. To avoid the ambiguity new set of features are generated by applying simple symmetric functions on each pair (e.g. $\{sum(a, c), sum(b, d)\}$). (b) The input edge feature is a 5- dimensional vector for every edge: the dihedral angle, two inner angles and two edge-length ratios for each face. The edge ratio is between the length of the edge and the perpendicular (dotted) line for each adjacent face. In MeshCNN they sort each of the two face-based features (inner angles and edge-length ratios), thereby resolving the ordering ambiguity and guaranteeing invariance. The images are from [Han+19].

algorithm. They introduce Graph Attention Layers, the building blocks of Graph Attention Networks.

Graph Attention Layer calculates the weights assigned to the neighboring features by calculating attention weights α_{ij} between the root node and its neighbors. These attention weights are calculated using attention coefficients which are calculated by

$$e_{ij} = a(W.h_i, W.h_j) \quad (3.3)$$

where $h_i \in \mathbb{R}^F$ is the input features for node i , $W \in \mathbb{R}^{F' \times F}$ is a linear transformation applied on input nodes and $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \rightarrow \mathbb{R}$ is a shared attention mechanism (F is the number of features in each node and F' is the number of output features).

Attention coefficients e_{ij} indicate the importance of node j 's features to node i . Attention coefficients are only calculated for nodes $j \in \mathcal{N}_i$ where \mathcal{N}_i is the neighborhood of node i

in the graph. To calculate the attention weights are normalized by applying softmax function on attention coefficients to make them easily comparable across different nodes as shown in the equation 3.4

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \quad (3.4)$$

In GAT, the attention mechanism a is a single-layer feedforward neural network parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, and applying the LeakyReLU nonlinearity. With this attention mechanism the attention weights the equation 3.4 is transformed into

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\vec{a}^T [W.h_i \parallel W.h_j]))}{\sum_{k \in \mathcal{N}_i} \exp(\text{LeakyReLU}(\vec{a}^T [W.h_i \parallel W.h_k]))} \quad (3.5)$$

where \parallel is the concatenation operation.

Once the attention weights are calculated the final representation h'_i of the node i , is calculated as follows

$$h'_i = \sigma \left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} W h_j \right) \quad (3.6)$$

They also extend their algorithm to employ multi-head attention, like in [Vas+17]. Multi-head attention uses a fixed number of independent attention mechanisms/heads and then at the end the outputs are concatenated. Multi-Head attention is discussed in detail in 4.2.

To employ Multi-Head attention with K heads the equation 3.7 is transformed into

$$h'_i = \parallel_{k=1}^K \sigma \left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k W^k \vec{h}_j \right) \quad (3.7)$$

where \parallel represents concatenation, α_{ij}^k are normalized attention coefficients computed by the k -th attention head (a^k), and W^k is the corresponding input linear transformation's weight matrix. Figure 3.4 shows graphical representation for the calculation of the attention weights and multi-head attention mechanism for graphs.

The approaches proposed in this thesis (discussed in detail in 5) uses attention mechanism and mutli-head attentions similar to Graph Attention Network. The difference is that in GAT the neighboring nodes are attended only by the root node which means that the attention weights are calculated only between the root node and the neighboring nodes. In our approach, every node attends to every other node which means attentions coefficients are calculated between every pair of nodes from the neighboring nodes and the root node.

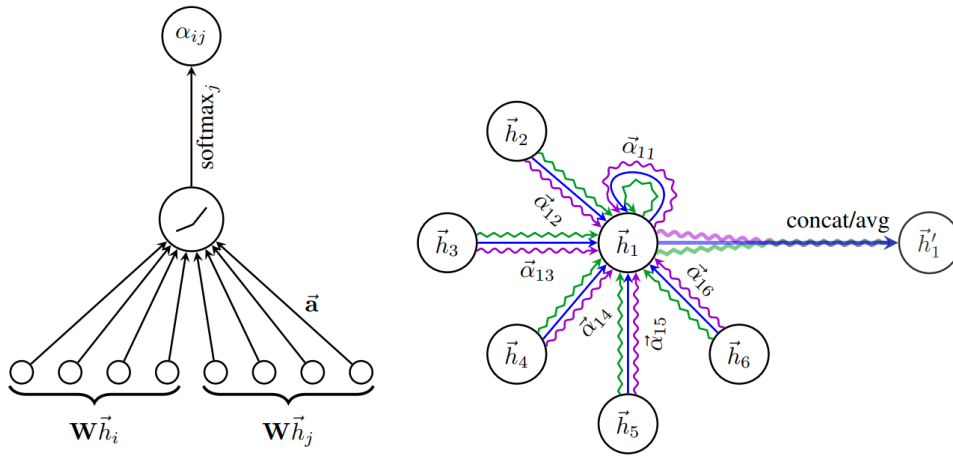


Figure 3.4: Left: The attention mechanism $a(W.h_i, W.h_j)$ employed by GAT, parametrized by a weight vector, applying a LeakyReLU activation. Right: An illustration of multihead attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain h'_1 .

4 Transformers

Transformer by Vaswani et al. [Vas+17] is a Deep Learning model introduced in 2017, used mainly in the field of Natural Language Processing. Transformers are designed to operate on sequential data to like other Recurrent Neural Networks (Long-short Term Memory (LSTM) [HS97], Gated Recurrent Units (GRU) [Chu+14] etc.), but work quite differently compared to the Recurrent Neural Network. Unlike in Recurrent Neural Network (RNN), Transformers do not require that the sequential data be processed in a specified order. This design paradigm allows for much more parallelization than other Recurrent Neural Network (RNN)s and therefore reduce training times.

Attention and Transformer Networks are an integral part of our ConvGNN algorithm which is discussed in detail in chapter 5. This section first covers what attention is and its various forms and how it originated and then details about transformer networks and various components involved in a transformer network such as Multi-Head Attention, Encoder, Decoder and Positional Encoding are covered.

4.1 Seq2Seq

Sequence-to-Sequence (Seq2Seq) models first introduced by Google in [SVL14] [Cho+14] are deep learning models that have achieved a lot of success in the tasks like machine translation, text summarization, speech recognition video/image captioning and are a backbone to numerous applications like Google Translate, voice-enabled devices, and online chatbots. Seq2Seq models are models that take a sequence of items and outputs another sequence of items. For tasks such as machine translation, the input is a sequence of words and output is the translated sequence of words.

A simple Seq2Seq model consists of an encoder and a decoder, where the encoder stores the information about the input sequence $\{x_1, x_2, \dots, x_n\}$ in the form of a hidden state vectors $\{h_1, h_2, \dots, h_n\}$ in order to help decoder make accurate predictions $\{y_1, y_2, \dots, y_m\}$. Then the decoder uses this hidden state vector to generate an output sequence. A typical

Seq2Seq models include some form of Recurrent Neural Networks in their encoders and decoders.

Figure 4.1 shows an example of a Seq2Seq model where the encoder and the decoder is a sequence of RNNs. The encoder consisting of a stack of RNNs takes the sequence as an input and generates a final hidden state vector which is then sent to the decoder which again consists of a stack of RNNs to predict an output sequence.

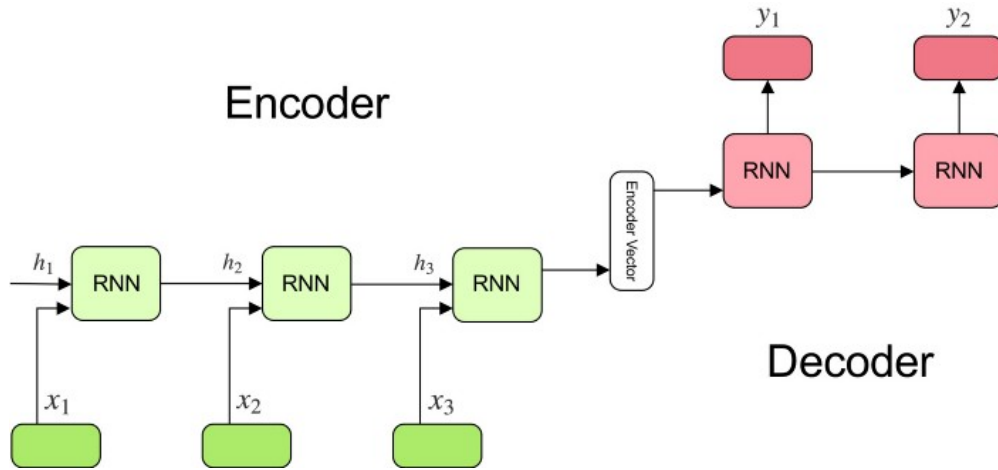


Figure 4.1: Sequence-to-sequence encoder decoder architecture, where the encoder and the decoder is a sequence of RNNs. The encoder consisting of stack of RNNs takes the sequence as an input and generates a final hidden state vector which is then sent to the decoder which again consists of stack of RNNs to predict an output sequence. The image is from [Kos19].

4.2 Attention

While simple Seq2Seq models have achieved a lot of success they have their drawbacks. As discussed above the job of an encoder is to generate a hidden state vector that stores all the context about the input sequence to help decoder predict an accurate output sequence. Hence the output sequence heavily relies on the hidden state generated by the encoder, making it very difficult for the model to deal with long sequences, where there is a high chance that information about the first few inputs have been lost by the end of the sequence.

Seq2Seq models in [LPM15] and [BCB16] solve the issue of loss of information by passing all the hidden input states generated by the encoder to the decoder and letting the

decoder decide at every step, which hidden states are most important. This is done by creating a context vector c_i which is a weighted average of all the hidden states provided by the encoder which can be calculated by the equation 4.1

$$c_i = \sum_{j=1}^N \alpha_{ij} h_j \quad (4.1)$$

Where, N is the length of the input sequence, h_j are the hidden state vectors generated by the encoder and α_{ij} are the weights assigned to the hidden state vectors at step i . Once the context vector is calculated, it is combined with the hidden state vector of the decoder by concatenation, thus forming a new attention hidden vector which is used by the decoder for predicting the output at that time instance. The weights α_{ij} are the amount of *attention* the i -th output should pay to the j -th input hidden state vector. Then the weights are computed by taking *softmax* over the attention scores, denoted by e , of the inputs with respect to the i -th output. The equation to calculate the weights α_{ij} is

$$\alpha_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k=1}^N \exp(e_{ik})} \quad (4.2)$$

where

$$e_{ij} = a(s_{i-1}, h_j) \quad (4.3)$$

Here a is an alignment model which decides the impact each input hidden state vector has on the final context vector, s_{i-1} is the hidden state from the previous step. In [BCB16], the alignment model is approximated by a small neural network, thus allowing both the models (Seq2Seq and the alignment model) to be optimized together. Attention weights for an english to french translation example from [BCB16] can be seen in figure 4.2.

4.2.1 Generalized Attention

Given a query q and a set of key-value pairs (K, V) , attention can be generalized as computation of the weighted sum of values, where the weights are dependent on the query and the corresponding keys. Here, since the query decides the impact of each value in the output; it can be said that the query attends to the values. Generalized attention can be written as

$$A(q, K, V) = \sum_i \frac{\exp(e_{qk_i})}{\sum_j \exp(e_{qk_j})} v_i \quad (4.4)$$

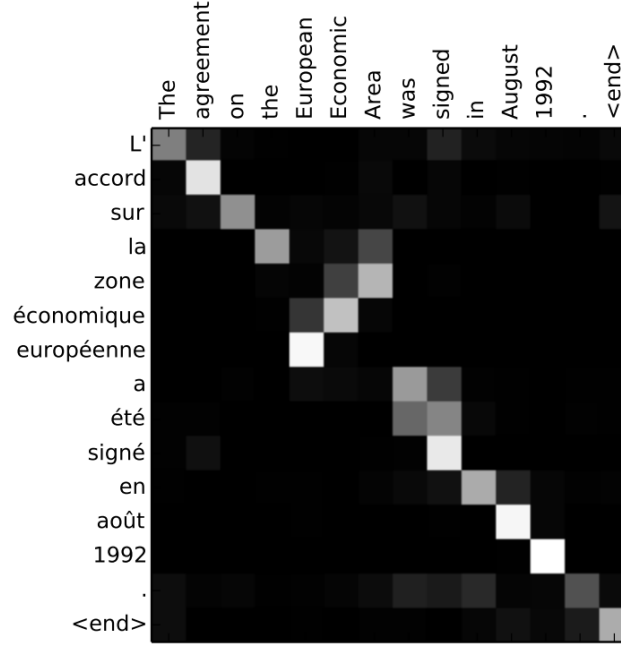


Figure 4.2: From [BCB16]. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight α_{ti} of the annotation of the j -th source word for the i -th target word in grayscale.

where $q \in \mathbb{R}^{d_q}$, $k_i \in \mathbb{R}^{d_k}$ and $v_i \in \mathbb{R}^{d_v}$ ($d_q, d_k, d_v \in \mathbb{N}$) are i -th key and value respectively and $e_j = a(q, k_j) \in \mathbb{R}$. For the Seq2Seq discussed above the query q is the previous hidden states s_{i-1} and the input hidden state vectors $\{h_0, \dots, h_n\}$ represent both the keys and values.

4.2.2 Alignment Model

The alignment model (a) comes in various forms. Section 4.1 showed an example where the alignment model a is approximated by a neural network that can be trained along with the Seq2Seq model. Below are a few more examples of alignment models that are quite frequently used in the research field.

Dot-Product Attention

Dot-Product attention as the name suggests is a simple dot product of the query and the key defined by

$$e_i = q^T k_i \quad (4.5)$$

where $d_k = d_q$.

Multiplicative Attention

Multiplicative attention is a special case of a dot product attention, where a linear transformation is applied to the key before the attention computation

$$e_i = q^T W k_i \quad (4.6)$$

where $W \in \mathbb{R}^{d_q \times d_k}$ is the linear transformation

Additive Attention

Additive attention introduced in [BCB16] is defined as

$$e_i = v^T \tanh(W_1 k_i + W_2 q) \quad (4.7)$$

where $W_1 \in \mathbb{R}^{d_3 \times d_k}$, $W_2 \in \mathbb{R}^{d_3 \times d_q}$ ($d_3 \in \mathbb{N}$) are linear transformations applied to the key and the query respectively and $v \in \mathbb{R}^{d_3}$ is a weight vector.

Concat Attention

Concat attention introduced in [LPM15] is similar to additive attention where instead of addition concatenation is used.

$$e_i = v^T \tanh(W(k_i \parallel q)) \quad (4.8)$$

where \parallel is a concatenation operation, $W \in \mathbb{R}^{(d_q+d_k)}$ is a linear transformation applied to the concatenated value and $v \in \mathbb{R}^{d_k+d_q}$ is the weight vector.

4.3 Transformer Network

Transformer Network like other Seq2Seq models have an encoder-decoder structure. Here, the encoder maps a sequence of input representations $x = (x_1, \dots, x_n) \in \mathbb{R}^{n \times d_{model}}$ to a sequences

of latent representations $z = (z_1, \dots, z_n) \in \mathbb{R}^{n \times d_{model}}$. Given the latent representation z and the masked output sequence $y_{mask} \in \mathbb{R}^{m \times d_{model}}$ of the actual output sequence $y = (y_1, \dots, y_m) \in \mathbb{R}^{m \times d_{model}}$ (in detail in section 4.3.2) the decoder predicts the probability y' of the next output. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next. A high level architecture of the transformer network is shown in the figure figure 4.3

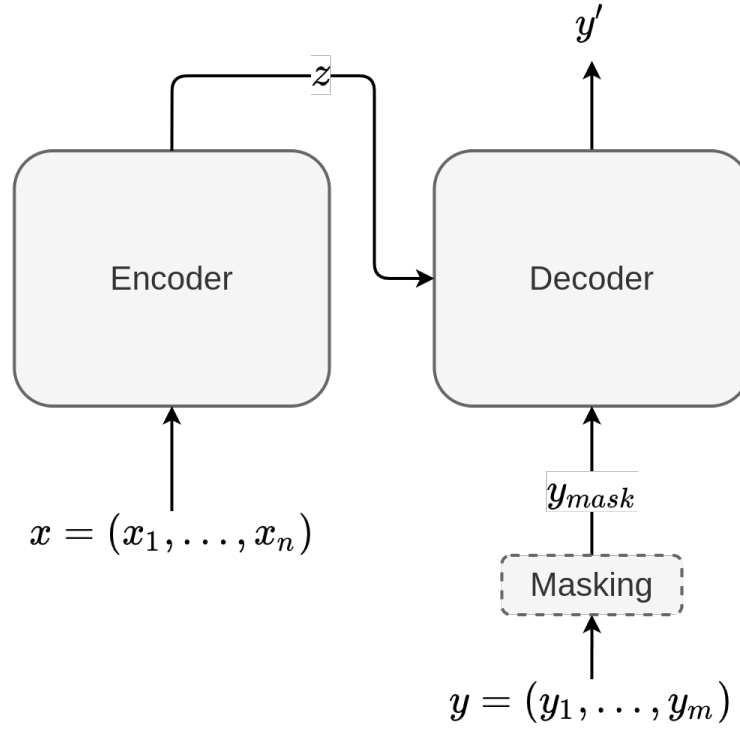


Figure 4.3: High level architecture of the transformer network.

In this section details about the components of transformer networks namely the encoder, the decoder, scaled dot-product attention, multi-head attention, self-attention, position-wise feed-forward networks and positional encoding are discussed. All these components and their underlying concepts play an important role in our methods. In chapter 5, how these components and concepts are used to build up our work is discussed.

4.3.1 Encoder

Encoder is a stack of N identical encoder layers (ENC). Each encoder layer consists of two sub-layers. The first layer the multi-head self-attention layer, and the second is a position-wise fully-connected feed-forward network. Each of the two sub-layers also consists of a residual connection [He+15], followed by layer normalization [BKH16].

The output of an encoder z for the sequence of input representations $x = (x_1, \dots, x_n)$ is computed by

$$z = ENC(x) = ENC_N(ENC_{N-1}(\dots(ENC_1(x + PE))\dots)) \quad (4.9)$$

where

$$ENC_i(x) = LayerNorm(SubLayer_i^1(x) + FFN^i(SubLayer_i^1(x))) \quad (4.10a)$$

$$SubLayer_i^1(x) = LayerNorm(x + MH_i(x, x, x)) \quad (4.10b)$$

where, $LayerNorm$ is the layer normalization operation, FFN is a position-wise fully connected feed-forward network, MH is a multi-head attention layer (section 4.3.5) and PE is the positional encoding (section 4.3.3)

4.3.2 Decoder

Decoder is also a stack of N identical decoder layers (DEC). Each decoder layer consists of three sub-layers. The first and the last sub-layers are identical to the first and the second sub-layers from the encoder layer. The middle sub-layer is a multi-head attention layer over the output of the encoder and the output of the first sub-layer in the decoder layer. Similar to encoder layers, decoder layers consist of residual connections for each of the three sub-layers, followed by layer normalization. The first multi-head self-attention layer in the decoder stack is modified to prevent positions from attending to subsequent positions. This masking $y_{mask} \in \mathbb{R}^{n \times d_{model}}$, combined with the fact that the output representations y are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i .

The output of an decoder $y' \in \mathbb{R}^{m \times C}$ (where C is the number of output classes) for the latent sequence $z = (z_1, \dots, z_n)$ and the masked output sequence y_{mask} is computed by

$$y' = DEC(z, y_{mask}) = DEC_N(DEC_{N-1}(\dots(DEC_1(z, y_{mask} + PE))\dots)) \quad (4.11)$$

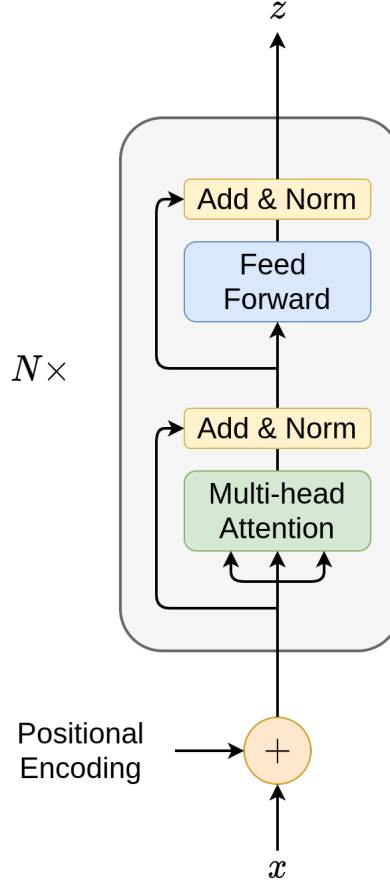


Figure 4.4: Architecture of the encoder in the transformer network.

where

$$DEC_i(z, y) = LayerNorm(SubLayer_i^2(z, y) + FFN_i(SubLayer_i^2(z, y))) \quad (4.12a)$$

$$SubLayer_i^2(z, y) = LayerNorm(SubLayer_i^1(y) + MH_i(SubLayer_i^1(y), z, z)) \quad (4.12b)$$

$$SubLayer_i^1(y) = LayerNorm(y + MH_i(y, y, y)) \quad (4.12c)$$

Where $LayerNorm$ is the Layer normalization operation, FFN is a position-wise fully connected feed-forward network, MH is a multi-head attention layer (section 4.3.5) and PE is the positional encoding (section 4.3.3)

Figure 4.4 and figure 4.5 shows the architecture of the encoder and decoder components of the transformer network. Figure 4.6 shows the entire architecture of the transformer network.

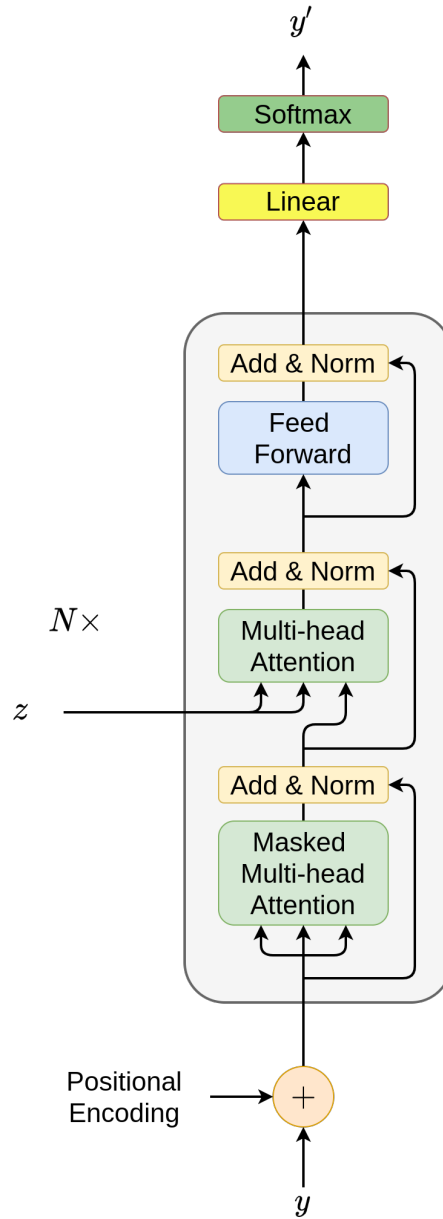


Figure 4.5: Architecture of the decoder in the transformer network.

4.3.3 Positional Encoding

As discussed earlier unlike RNN transformers by design do not need sequential data as there is no recurrence. In order for the model to make use of the order of the sequence, [Vas+17] add extra information about the relative or absolute position of the tokens in the sequence.

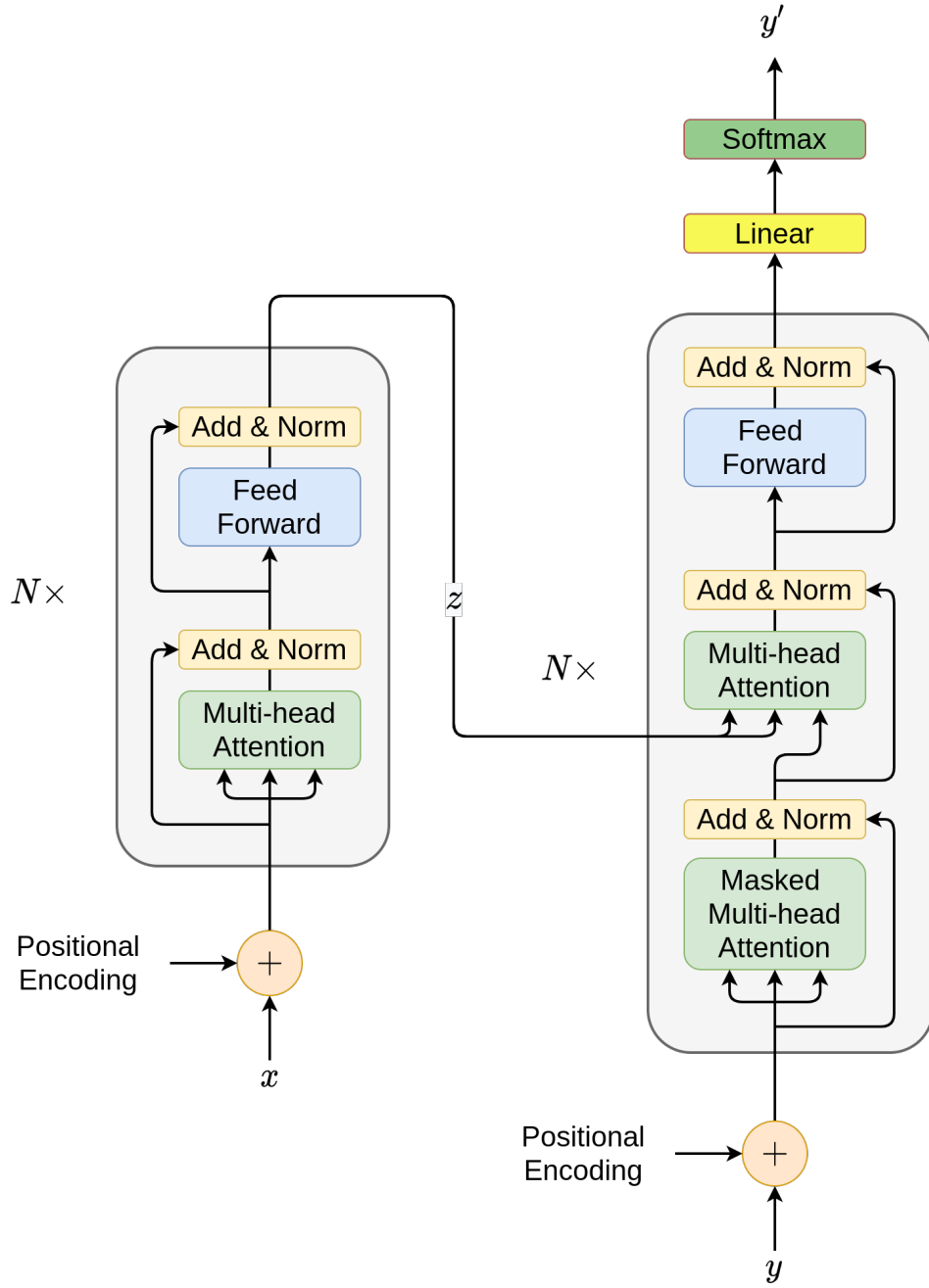


Figure 4.6: Low level architecture of the complete transformer network.

To do this the authors propose positional encodings which is a function of the position pos of the token and depth i in the feature dimension. These positional encodings have the same

dimension as the input representations and are summed to the input representations at the bottoms of the encoder and decoder stacks. Transformers use sine and cosine functions of different frequencies to generate positional encodings as shown below

$$PE(pos, i) = \begin{cases} \sin(pos/10000^{2i/d_{model}}) & \text{if } i \text{ is even} \\ \cos(pos/10000^{2i/d_{model}}) & \text{if } i \text{ is odd} \end{cases} \quad (4.13)$$

where d_{model} is the number of features in the input. Each dimension of the postional encoding correspond to a sinusoid. Authors claim that this would allow the model to easily learn to attend by relative positions, since for any fixed offset k , PE_{pos+k} can be represented as a linear function of PE_{pos} .

Postional encodings are also used in our transformer based convolution layer (discussed in section 5.3) and while applying atrous convolution (discussed in section 5.5).

4.3.4 Scaled Dot Product Attention

Additive attention (equation 4.7) and dot-product attention (equation 4.5) are similar in theoretical complexity, but dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code. Authors in [Vas+17] claims that with an increasing number of features (d_q, d_k) in the query q and keys k , additive attention outperforms dot-product attention.

Scaled dot product attention is an attention mechanism introduced by Google in [Vas+17] designed to have the same computation efficiency as a dot product attention but also have the peformance of an additive attention. For a given query $q \in \mathbb{R}^{d_{qk}}$ and a key $k \in \mathbb{R}^{d_{qk}}$ scaled dot-product attention computes the attention scores by:

$$e_i = \frac{q^T k}{\sqrt{d_{qk}}} \quad (4.14)$$

Scaled dot-product attention is identical to the dot-product attention discussed above except for the scaling factor $\sqrt{d_{qk}}$. An intuitive explanation is that for large values of d_{qk} , the dot product grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. The scaling factor should help the softmax values stay close to zero and thus have good gradients to be back propagated.

Hence for a given set of queries $Q \in \mathbb{R}^{N \times d_{qk}}$ and key-value pairs $K \in \mathbb{R}^{M \times d_{qk}}, V \in$

$\mathbb{R}^{M \times d_v}$ the attention (Figure 4.7) $A(Q, K, V) \in \mathbb{R}^{N \times d_v}$ is computed as

$$A(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_{qk}}} \right) V \quad (4.15)$$

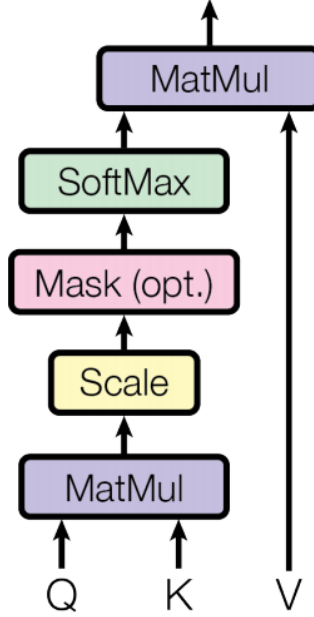


Figure 4.7: Scaled Dot-Product Attention over $Q \in \mathbb{R}^{N \times d_{qk}}$ and key-value pairs $K \in \mathbb{R}^{M \times d_{qk}}, V \in \mathbb{R}^{M \times d_v}$.

4.3.5 Multi-Head Attention

Multi-Head attention also introduced in [Vas+17] unlike scaled dot-product attention, instead of performing a single attention function with d_q, d_k, d_v dimensional query, key and values respectively, computes h attentions in parallel (called heads) over h linearly projected queries, keys and values with different, learned linear projections to d'_q, d'_k , and d'_v dimensions respectively. The outputs from these h attentions are then concatenated and once again projected, resulting in the final values.

For $d_q = d'_q * h$, while the computation cost remains the same, author in [Vas+17] claims that having multiple attention heads allows the model to learn relevant information in different child spaces and thus yielding better performance than having a single attention mechanism.

Hence for a given set of queries $Q \in \mathbb{R}^{N \times d_{qk}}$ and key-value pairs $K \in \mathbb{R}^{M \times d_{qk}}, V \in \mathbb{R}^{M \times d_v}$ the mutli-head attention (Figure 4.8) $MH(Q, K, V) \in \mathbb{R}^{N \times d_v}$ is computed as

$$MH(Q, K, V) = (head_1 \parallel \dots \parallel head_h)W_0 \quad (4.16)$$

where $head_i = A(QW_i^Q, KW_i^K, VW_i^V)$ and the matrices $W_i^Q \in \mathbb{R}^{d_q \times d'_q}, W_i^K \in \mathbb{R}^{d_k \times d'_k}, W_i^V \in \mathbb{R}^{d_v \times d'_v}$ are projection parameters.

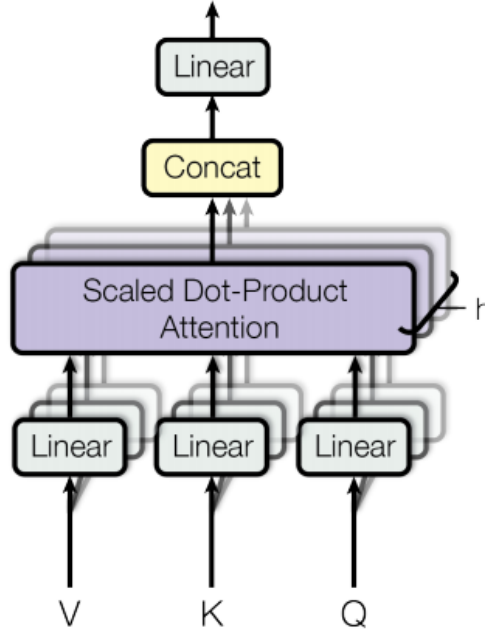


Figure 4.8: Multi-head attention layer from the transformer network over $Q \in \mathbb{R}^{N \times d_{qk}}$ and key-value pairs $K \in \mathbb{R}^{M \times d_{qk}}, V \in \mathbb{R}^{M \times d_v}$.

4.3.6 Self Attention

Self Attention is a special form of Attention introduced in [Vas+17] where all the queries, keys, and values are the same ($Q = K = V$). The goal of the self-attention is to learn the dependencies among inputs and use that information to capture the internal structure of the inputs.

Self Attention was introduced as an alternative to recurrent and convolutional layers which are commonly used for mapping one variable-length sequence of representations to another sequence of equal length, to reduce the computational complexity per layer, increase the

amount of computation that can be parallelized while having a similar performance.

$$SA(X) = A(X, X, X) = \text{softmax} \left(\frac{XX^T}{\sqrt{d_x}} \right) X \quad (4.17)$$

where $X \in \mathbb{R}^{N \times d_x}$

As discussed in the sections 4.3.1 and 4.3.2 the encoder and decoder components of the transformers uses multi-head attention in three different ways

- Encoder-decoder attention in the decoder layers. In this the queries come from the previous layer in the decoder stack and the keys and values come from the output of the encoder stack. This allows every position in the decoder to attend over all positions in the input sequence. This attention is similar to encoder-decoder attention mechanism in Seq2Seq models discussed in section 4.1.
- Encoder self-attention in the encoder layers. In this all of the queries, keys and values come from the output of the previous layer in the encoder stack. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Decoder self-attention in the decoder layers. Similar to self-attention layers in the encoder all of the queries, keys, and values come from the output of the previous layer in the decoder stack. These attention layers allow each position to attend to all positions in the decode up to and including that position. This is implemented inside of the scaled dot-product attention by masking out all values in the input of the softmax which correspond to illegal connections.

4.3.7 Position-wise Feed-Forward Networks

A Position-wise Feed-Forward network consists of two linear transformations with a ReLU activation in between as is referred to as Feed Forward in the following sections. For $x \in \mathbb{R}^{N \times d_x}$

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (4.18)$$

where $W_1 \in \mathbb{R}^{d_x \times d_f}$, $W_2 \in \mathbb{R}^{d_f \times d_x}$, $b_1 \in \mathbb{R}^{d_f}$ and $b_2 \in \mathbb{R}^{d_x}$

4.4 Examples

In this section an example of an application of transformers for Machine Translation task and application of attentions in Convolutional Graph Neural Network is discussed.

4.4.1 Machine Translation

As discussed in the beginning of chapter 4 and in section 4.1 Sequence-to-Sequence models, in particular Seq2Seq models with attention such as transformer network became very famous for its success in Natural Language Processing tasks especially Machine Translation Task.

At a high-level Machine Translation is the task of converting a sequence of words from one language to a sequence of words in another language (figure 4.9). For example, converting the sentence (sequence of words) "How are you?" in English to "¿Cómo estás?" sentence in Spanish.



Figure 4.9: Machine translation task at high level.

Most of the Sequence-to-Sequence models are auto-regressive i.e. at each step it consumes information about the previously generated output of the model (acts as a conditioning on the output of the model) additional to the regular input to the model. For example for the task of translating the sentence "How are you?" to "¿Cómo estás?" a Seq2Seq with self-attention model takes three steps to generate the translated sentence. Following are the details of each step

- In the first step the input to the model is a sequence of words ["How", "are", "you", "?"] and "<s>", where "<s>" points to the start of the sentence and acts as a dummy for the "previous output" discussed earlier (in practice these words are split into tokens and these tokens have embeddings, for the scope of this thesis is it assumed that the sequence of words is the input) and the model is expected to predict the word "¿Cómo" as the next word.

$$\begin{aligned}
 p(w_i | ["How", "are", "you", "?"], ["<s>"]) \\
 = DEC(ENC(["How", "are", "you", "?"], "<s>"))[i]
 \end{aligned}
 \tag{4.19}$$

Where, w_i is a word from the spanish vocabulary.

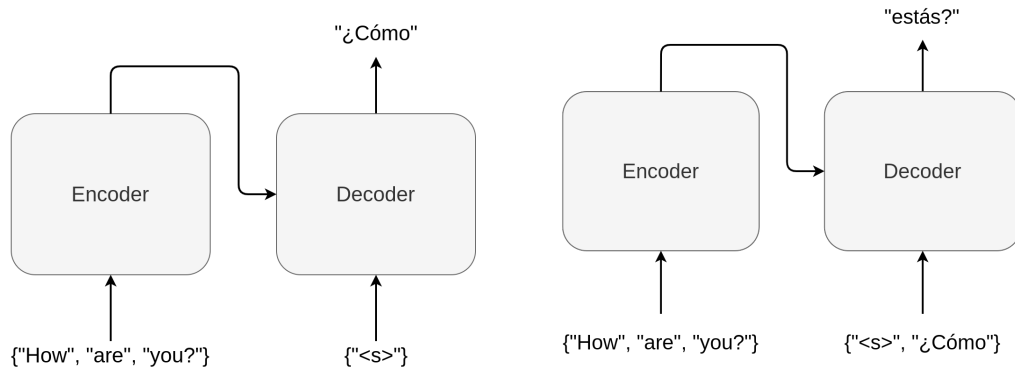
- In the second step the input is now the sequence of words ["How", "are", "you", "?"] and the sequence of previously generated outputs ["<s>", "¿Cómo"], and the model is expected to predict the word "estás?" as the next word.

$$\begin{aligned}
 p(w_i | ["How", "are", "you", "?"], ["<s>", "¿Cómo"]) \\
 = DEC(ENC(["How", "are", "you", "?"], ["<s>", "¿Cómo"])[i]
 \end{aligned}
 \tag{4.20}$$

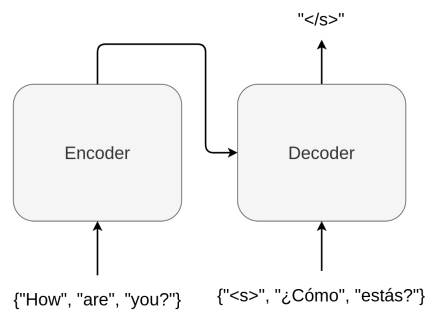
Where, w_i is a word from the spanish vocabulary.

- Similar to the second step the input is the sequence of words ["How", "are", "you", "?"] and the sequence of previously generated outputs ["<s>", "¿Cómo", "estás?"] and the model is expected to predict the word "</s>" which refers to the end of the sentence.

Figure 4.10 shows the details of each of the steps discussed above using the transformer network.



(a) Step one of the translation using transformer . (b) Second step of the translation using transformer



(c) Final step of the translation using transformer .

Figure 4.10: Shows inputs to each of the step involved in translating the sentence "How are you?" to spanish. Demonstrates the auto-regressive design of the model.

5 Our Approach

Transformer networks and their components form the core to our work in this thesis. In this section, two approaches to the task of 3D Mesh Segmentation that are based on transformer network and its components are discussed. Our first approach shows how a Mesh Segmentation task can be treated as a Machine Translation Task, thus how transformer networks can be modified to be used for Mesh Segmentation. Our second approach is to design a Graph Convolution layer called Transformer Convolution based on the various components of the transformer discussed in chapter 4 that can be used in any Convolutional Graph Neural Network.

Following these, two extensions to the approaches are discussed. First how, where and why is the positional encoding (discussed in section 4.3.3) useful for Transformer Convolution (TransConv) is discussed. Then the second extension, the application of atrous convolution to a graph and how positional encoding can be incorporated into atrous convolution is discussed.

5.1 Problem Statement Recap

To recall the problem statement as discussed in chapter 1, for any given Mesh or Graph $G : (X, E)$, where $X : \{x_1, \dots, x_N\}, x_i \in \mathbb{R}^F, F \in \mathbb{N}$ are the features of the vertices and $E : \{\dots, e_{ij}, \dots\}, e_{ij} \in \mathbb{R}^2$ are the edges between the vertices, to assign each of the vertices a label $Y : \{y_1, \dots, y_N\}, y_i \in \{0, \dots, C\}$ from a predefined set of labels of length C .

5.2 Transformer for Mesh Segmentation (MeshTrans)

Previously in section 4.4.1 an example of transformer network being used for the task of Machine Translation was shown. In this section, our first approach on how transformers can similarly be used for the task of mesh segmentation is discussed.

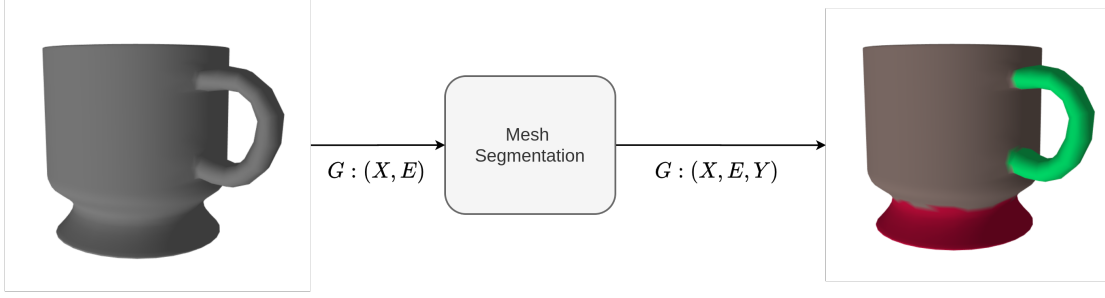


Figure 5.1: Shows the graphical representation of the problem statement for this thesis, which is semantic segmentation of 3D meshes. The input on the left hand side is a 3D mesh and the expected output is the semantic segmentation of the 3D mesh.

At a high-level, mesh segmentation can also be thought of as a Machine Translation task where the source vocabulary is the set of all vertices in the mesh G and the target vocabulary is the set of all labels for the mesh. Then the task of segmenting a single vertex is equivalent to translating a single sentence with the input sentence now being a set of features of neighboring vertices $\{\{x_j : j \in \mathcal{N}_i\} \cup x_i\}$ instead of ["How", "are", "you", "?"]. The conditioning on the decoder is now the features of the root vertex x_i instead of "<s>". The output here is the probability y_i of the label from the target category associated to the root vertex i instead of the probability of the next word in the Spanish vocabulary.

In a Machine Translation task, the encoder has the responsibility to store information about the words in the source language that helps decoder to accurately predict the next word in the target language, similarly in our approach the encoder has the responsibility to store information about the neighborhood of the root vertex that can help the decoder to make accurate predictions about the label of the root vertex.

Also, transformers by design are invariant to permutation, which is also the reason why positional encoding is added for the tasks that have order in input as shown in section 4.3.3. This permutation invariance makes them much more suitable for processing graph data structures like meshes. Hence in our first approach, a transformer architecture is used as our baseline and modify it for the task of Mesh Segmentation.

In our MeshTrans, for a mesh $G : (X, E)$ to calculate the label for a vertex i using a transformer network, the input X_i to the encoder will be the union of the set of features of the neighboring vertices $\mathcal{N}_{x_i} = \{x_j : j \in \mathcal{N}_i\} \in \mathbb{R}^{n \times F}$, (where $\mathcal{N}_i = \{j : e_{ij} \in E\}$ is the

neighborhood) and the features of the root vertex $\{x_i\}$, and the input to the decoder is the features of the output of the encoder Z_i and the input features of the root vertex x_i . The final output of the transformer network is the vector of probabilities of the label associated to the root vertex. The probability that the root vertex is assigned the label c_i is calculated as

$$p(c_j|X_i, \{x_i\}) = DEC(ENC(X_i, \{x_i\}))[j] \quad (5.1)$$

where $X_i = \{\mathcal{N}_{x_i} \cup \{x_i\}\}$, c_j is the j th label from the predefined set of labels C .

Figure 5.2 shows the final architecture of our first approach Transformer for Mesh Segmentation (MeshTrans). For a given graph $G : X, E$ and vertex i , a neighborhood matrix X_i is created, which is passed through a feed-forward neural network to increase the feature dimensions d_{model} before passing them to the encoder stack. Similarly, the input to the decoder is also upscaled using a feed-forward neural network as shown in the figure. Table 5.1 shows a list of hyper parameters for the MeshTrans network.

Table 5.1: Shows the list of hyper parameters and their descriptions in a MeshTrans network.

Hyper parameter	Description
num_layers	Number of channels in the input
heads	Number of heads to be used in the multi-head attention layer
d_{model}	Number of channels to which the input should be upscaled to before passing to the encoder
d_f	Number of output channels in the first feed-forward network of the position-wise feed-forward neural networks (discussed in section 4.3.7)

5.3 Transformer Convolution (TransConv)

In our first approach, the inputs to the transformer architecture are modified in such a way that a Mesh Segmentation task is treated as a Machine Translation Task. Such an approach is very closely linked to the model architecture and thus can not be treated as a convolutional layer that can fit into any model architecture.

Our second approach is to design a Graph Convolution layer called Transformer Convo-

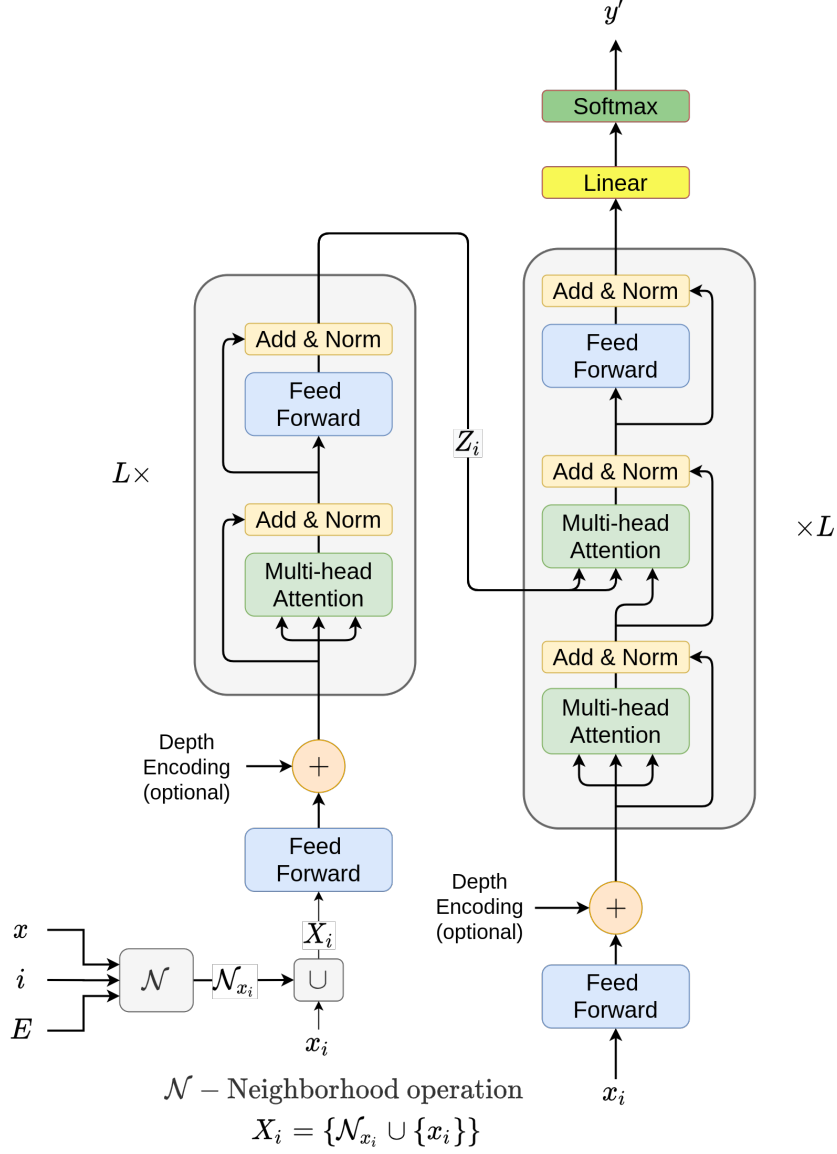


Figure 5.2: Architecture of Transformer for Mesh Segmentation, consisting of L encoder layers and L decoder layers. For any vertex i , a neighborhood matrix X_i is created, which is passed through a feed-forward neural network before passing them to the encoder stack. The encoder output along with the root node features are passed through the decoder stack to generate output category probabilities.

lution (TransConv) using the components of the transformer network that can be treated similar to any convolution operation in images. The advantage of such an operation is that it can be fit into any model architecture. A Graph Convolution operation like any convolution operation on images (with stride one) can be defined as follows

In general, for any mesh $G : (X, E)$ and vertex/node features $h_i, i \in [1, \dots, N], h_i \in \mathbb{R}^F, F \in \mathbb{N}$ at any point in the network, the output of the Graph Convolution(GC) $h'_i, i \in [1, \dots, N], h'_i \in \mathbb{R}^{F'}, F' \in \mathbb{N}$

$$h'_i = GC(h_i, E) \quad (5.2)$$

As discussed in chapter 3, the goal of any graph convolution operation is to gather information from the neighborhood and store important information in the output representations. In many attention based graph convolution operations such as Graph Attention Network [Vel+18], FeaStNet [VBV18], Dynamic Graph CNN [Wan+19], etc. the attention weights are calculated only between the root vertex and the neighboring vertices (i.e. root attends the neighbors). In our graph convolution layer Transformer Convolution (TransConv), along with the attention weights between the root vertex and the neighboring vertices, new attention weights between every possible pair of vertices from the set of neighboring vertices are also calculated (i.e. every neighbor attends every other neighbor).

To do this, our TransConv is split into two operations

- First operation is where each vertex in the neighborhood attends every other vertex in the neighborhood.
- Second operation is the attention based aggregation operation which aggregates all the information learnt by the neighboring vertices after the first operation.

The first operation is equivalent to the encoder layer (figure 5.3) operation in the encoder of a transformer followed by a simple feed-forward neural network. The input in this case is the union of the set of neighboring vertex features $\mathcal{N}_{h_i} = \{h_j : i \in \mathcal{N}_i\} \in \mathbb{R}^{n \times F}, n = |\mathcal{N}_i|$ and the features of the root vertex h_i and the output $Z_i \in \mathbb{R}^{(n+1) \times F}$ is a set of encoded representations which is of the same length as the input. The message passing MP operation is defined as

$$Z_i = MP(H_i) = FFN_1(ENC_0(H_i)) \quad (5.3a)$$

$$\text{where } ENC_0(H_i) = LN(SubLayer(H_i) + FFN_2(LN(H_i))), \quad (5.3b)$$

$$SubLayer(H_i) = LN(x + MH(H_i, H_i, H_i)), \quad (5.3c)$$

$H_i = \{\mathcal{N}_{h_i} \cup \{h_i\}\} \in \mathbb{R}^{(n+1) \times F}$, LN is a Layer Normalization operation, FFN_i s are a point-wise feed forward neural networks and MH is a multi-head attention layer.

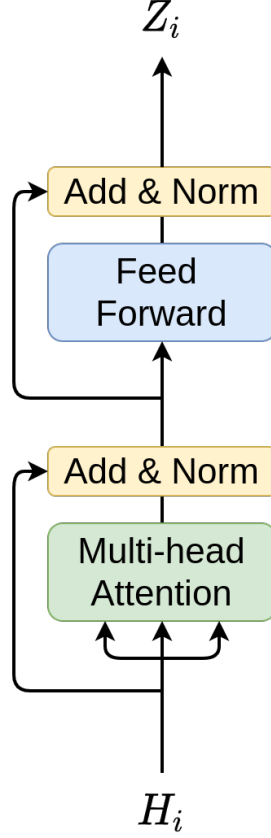


Figure 5.3: Encoder layer operation from the encoder part of the transformer network.

The second operation of our TransConv is a learning-based aggregation inspired by the Pooling by Multihead Attention block from Set Transformers [Lee+19] which is discussed in detail in the following section.

5.3.1 Set Transformer

Set Transformers [Lee+19] are an attention-based neural network that is designed to process data in the form of sets. In this section, details about the building blocks of the Set Transformers and how these blocks are used for our second step (aggregation) of our graph convolution TransConv are discussed.

Similar to the transformer architecture, the set transformer consists of an encoder and a

decoder. The encoder and decoder are made up of three blocks

- Multihead Attention Block (MAB)
- Set Attention Block (SAB)
- Pooling by Multihead Attention (PMA)

The Set Attention Block (SAB) is the same as an encoder layer in the encoder of a transformer network with a different name, hence not discussed in this section.

Multihead Attention Block (MAB)

In Set Transformers, MAB is very similar to an encoder layer in the encoder of the transformer network. The only difference is that MAB takes two sets of inputs S_1, S_2 instead of a single input and outputs a single set S' . The remaining components of the blocks are exactly like an encoder layer as shown in figure 5.4a. The first input acts as the query and the second acts as both the keys and values. Given two sets $S_1, S_2 \in \mathbb{R}^{n \times d}$, MAB is defined as

$$S' = MAB(S_1, S_2) = LayerNorm(H + FFN(H)) \quad (5.4)$$

where $H = LayerNorm(S_1 + MH(S_1, S_2, S_2))$, FFN is a feed-forward network, LayerNorm is a Layer Normalization operation and MH is a multihead attention.

Pooling by Multihead Attention (PMA)

In Set Transformers PMA blocks as the name suggests is used to pool a set, i.e. to reduce the size of the set with some form of aggregation. Instead of using aggregation schemes such as dimension wise average or maximum, PMA aggregates features by applying multihead attention on a learnable set of k seed vectors $U \in \mathbb{R}^{k \times d}$. For a given input set of features $S \in \mathbb{R}^{n \times d}$, PMA with k seed vectors is defined as

$$PMA_k(S) = MAB(U, FFN(S)) \quad (5.5)$$

where MAB is the Multihead Attention Block from equation 5.4 and FFN is a feed-forward neural network. The output set length is determined by the number of seed vectors used. Here, aggregation by attention is beneficial because the influence of each item in the set on aggregation is not necessarily equal.

Graphical representation of both Multihead Attention Block and Pooling by Multihead Attention are shown in the figure 5.4

As said earlier, the Pooling by Multihead Attention block is used for the second step of our graph convolution operation TransConv which aggregates the information learned by the root node and its neighbors after the first step of the convolution operation. Since after the graph convolution operation, a one to one mapping of the input features to the output

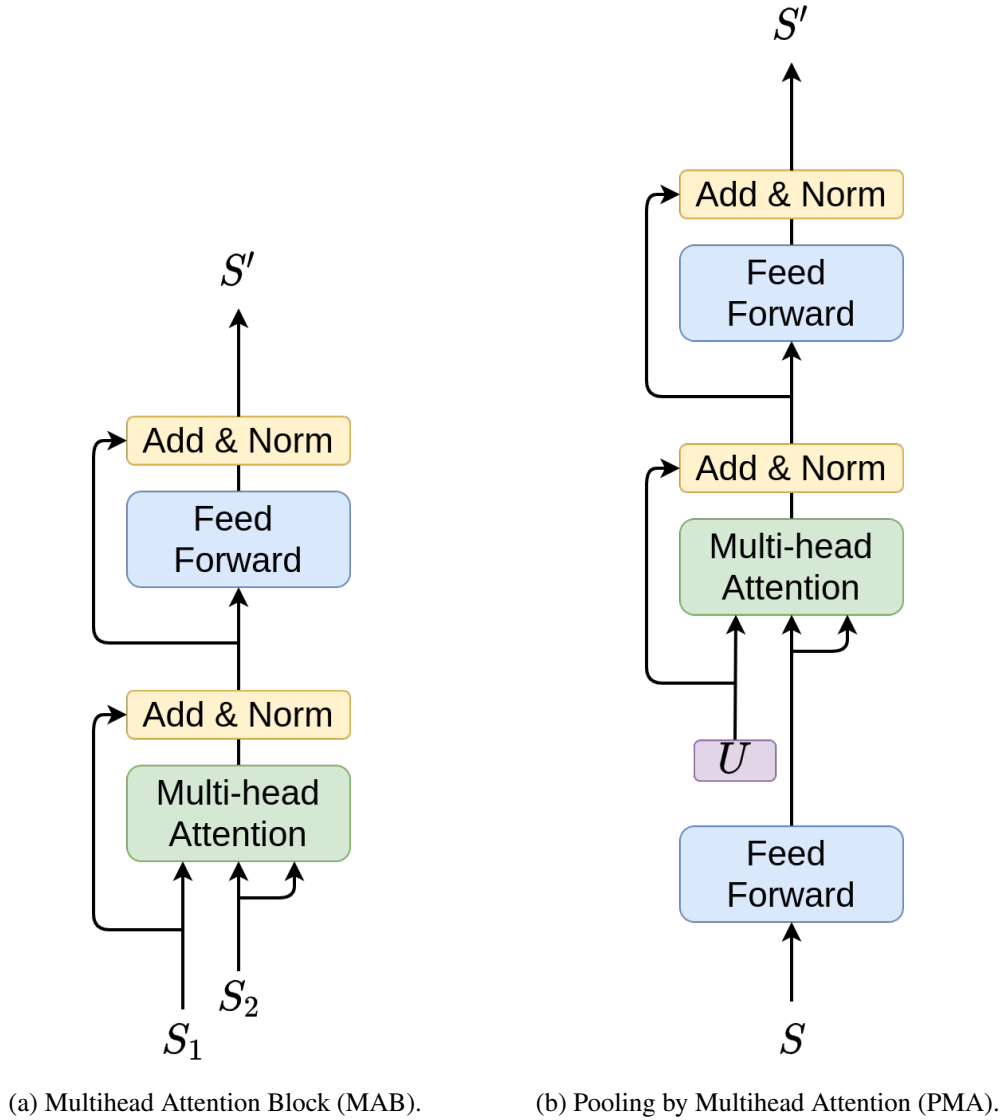


Figure 5.4: Two components of Set Transformers that are used in our TransConv operation.

features is needed, a single seed vector $u \in \mathbb{R}^{1 \times F'}$ is used in our second step. Hence for a set of input features $z_i \in \mathbb{R}^{(n+1) \times F'}$ the aggregation operation can be defined as

$$h'_i = PMA(Z_i) = MAB(u, FFN(Z_i)) \quad (5.6)$$

5.3.2 Final Architecture

In our second approach (TransConv), for any mesh $G : (X, E)$ and vertex/node features $h_i \in \mathbb{R}^F, i \in [1, \dots, N]$ at any point in the network, the output of a Transformer Convolution $h'_i \in \mathbb{R}^{F'}, i \in [1, \dots, N]$ is a combination of the first and the second operations and is defined as

$$h'_i = TransConv(h_i) = PMA(MP(H_i)) \quad (5.7)$$

where PMA and MP operations are from equations 5.6 and 5.3 respectively and $H_i = \{\mathcal{N}_{h_i} \cup \{h_i\}\} \in \mathbb{R}^{(n+1) \times F}, n = |\mathcal{N}_i|$.

Figure 5.5 shows the final architecture of our graph convolution layer Transformer Convolution layer. First, for any given vertex all the neighboring vertex features (found using the edges) are stacked to form a matrix $H_i \in \mathbb{R}^{(n+1) \times F}$. This matrix is passed through a feed-forward neural network to increase the feature dimensions to d_{model} . The output is then passed through the encoder layer followed by a simple feed-forward neural network to match the output channels F' . The output is passed on to the Pooling by Multihead Attention layer to generate the final output of the Convolution layer. Table 5.2 shows details about the hyper paramters in a single Transformer Convolution layer.

5.3.3 Support for Batching

It is quite common for vertices in graphs to have different degrees, to support batch operations through our graph convolution operation TransConv, the highest degree (n') among all the vertices in the batch is calculated and zero features are appended to the vertices with degree less than the maximum degree to form a tensor $\in \mathbb{R}^{N \times n' \times F}$ where, N is the number of vertices and F is the number of features for each vertex.

5.4 Depth Encoding

In section 4.3.3 it was shown how Positional encoding is beneficial in transformer network and how it is calculated. This section covers details about how and why can Positional

Encoding be useful for the task of Mesh Segmentation.

Having looked at the architecture of Transformer Convolution layer, one can observe that there is no way for the model in both the operations (message passing and learning-based aggregation) to know which vertex is the root vertex for which the segmentation needs to be calculated. To help the model learn which vertex is the root vertex depth encoding is added similar to positional encoding in transformer network to the input to differentiate neighboring

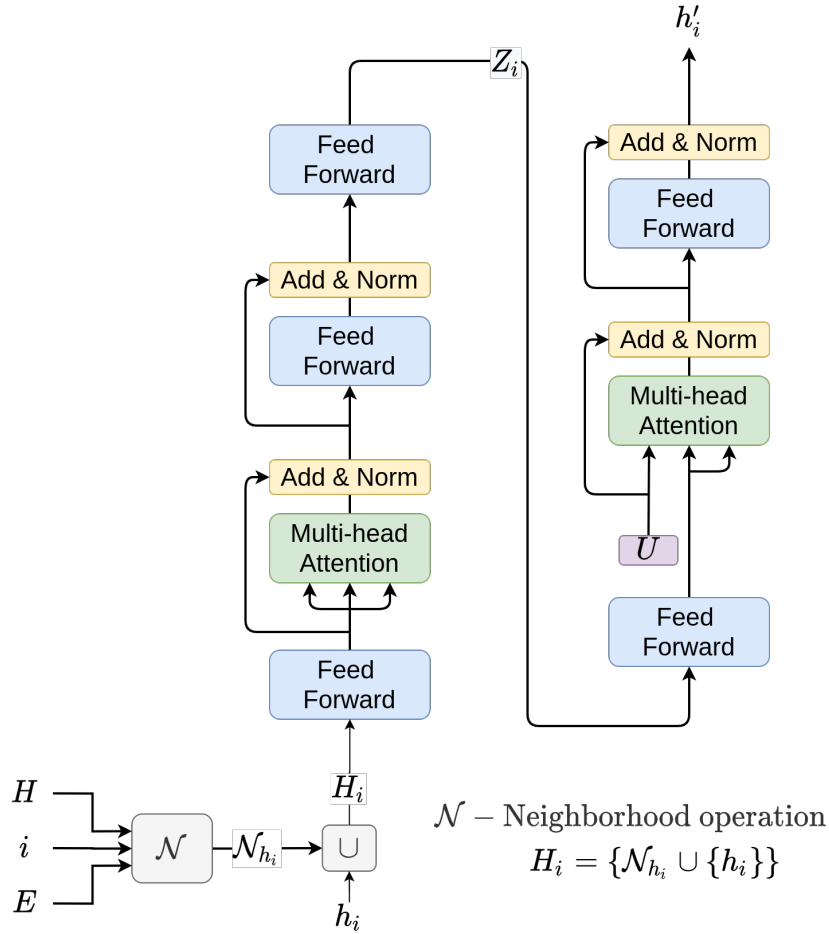


Figure 5.5: Final architecture of the Transformer Convolution layer. For any given vertex all the neighboring vertex features (found using the edges) are stacked to form a matrix $H_i \in \mathbb{R}^{(n+1) \times F}$. This matrix is first passed through a feed-forward neural network and then passed through the encoder layer followed by another simple feed-forward neural network. The output is passed on to the Pooling by Multihead Attention layer to generate the final output of the Convolution layer.

Table 5.2: Shows for each of the hyper parameters in a single Transformer Convolution layer, the names and descriptions.

Hyper parameter	Description
F'	Number of channels/features to be present in the output
heads	Number of heads to be used in the multi-head attention layer
d_{model}	Number of channels to which the input should be upscaled to before passing to the encoder (also called the depth of the layer)
d_f	Number of output channels in the first feed-forward network of the position-wise feed-forward neural networks (discussed in section 4.3.7)

vertices from the root vertex. Depth encoding is similar to positional encoding in transformer networks where the positions of each vertex is the depth (shortest path distance) from the root vertex. That is the root vertex is assigned a position of 0 and the neighboring vertices are assigned a value 1.

In our a scaled version of positional encoding function from transformer network is used as shown below

$$DE(depth, i) = \begin{cases} \alpha * \sin(depth/10000^{2i/d_{model}}) & \text{if } i \text{ is even} \\ \alpha * \cos(depth/10000^{2i/d_{model}}) & \text{if } i \text{ is odd} \end{cases} \quad (5.8)$$

where α is the scaling factor decided according the distribution of the input feature values, $depth$ is the depth of the neighbor vertex from the root vertex and $i \in [0, \dots, F]$ is the depth along the feature dimension. Like in transformer network the depth encoding is added to the input only once at the very beginning of the model before calling the TransConv layers.

5.5 Atrous Convolution

Atrous Convolution also known as dilated convolution is a type of convolution operations first introduced in [YK16] and became famous with the introduction of DeepLab [Che+17] model which produced state of the art results for the task of Semantic Segmentation of images.

The output $y[i, j]$ of an atrous convolution operation on a two dimensional input $x[i, j]$ with a filter $w[k, l]$ is defined as

$$y[i, j] = \sum_{k=1}^K \sum_{l=1}^L x[i + r.k, j + r.l] w[k, l] \quad (5.9)$$

where r is the rate parameter, which corresponds to the stride with which the input signal is sampled. Figure 5.8 shows graphical interpretation of the Atrous Convolution operation.

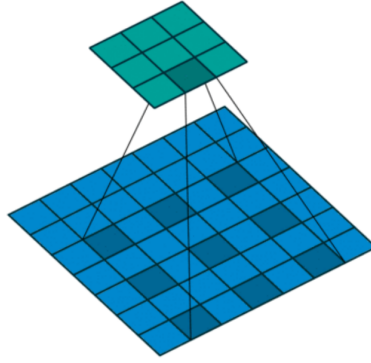


Figure 5.6: Atrous convolution on 2D input using a kernel of width size 3 with rate $r = 2$. The image is from [Prö18].

The main idea behind atrous convolution is to increase the receptive field without increasing the memory consumption and without any decrease in the spatial resolution. Figure 5.7 shows an example of exponentially increasing receptive field using atrous convolutions.

In our work, the concept of atrous convolutions is extended to Convolutional Graph Neural Network. Atrous convolution is applied to graphs by updating edge matrix E of a graph with new edges. These new edges correspond to the vertex pairs whose geodesic distance (distance along the initial graph edges) is less than a predefined depth. The new edges are found by performing a Breadth-First Search (BFS) for each of the vertices on the graph with a limit on the depth of traversal. Once all the new edges with depth less than the

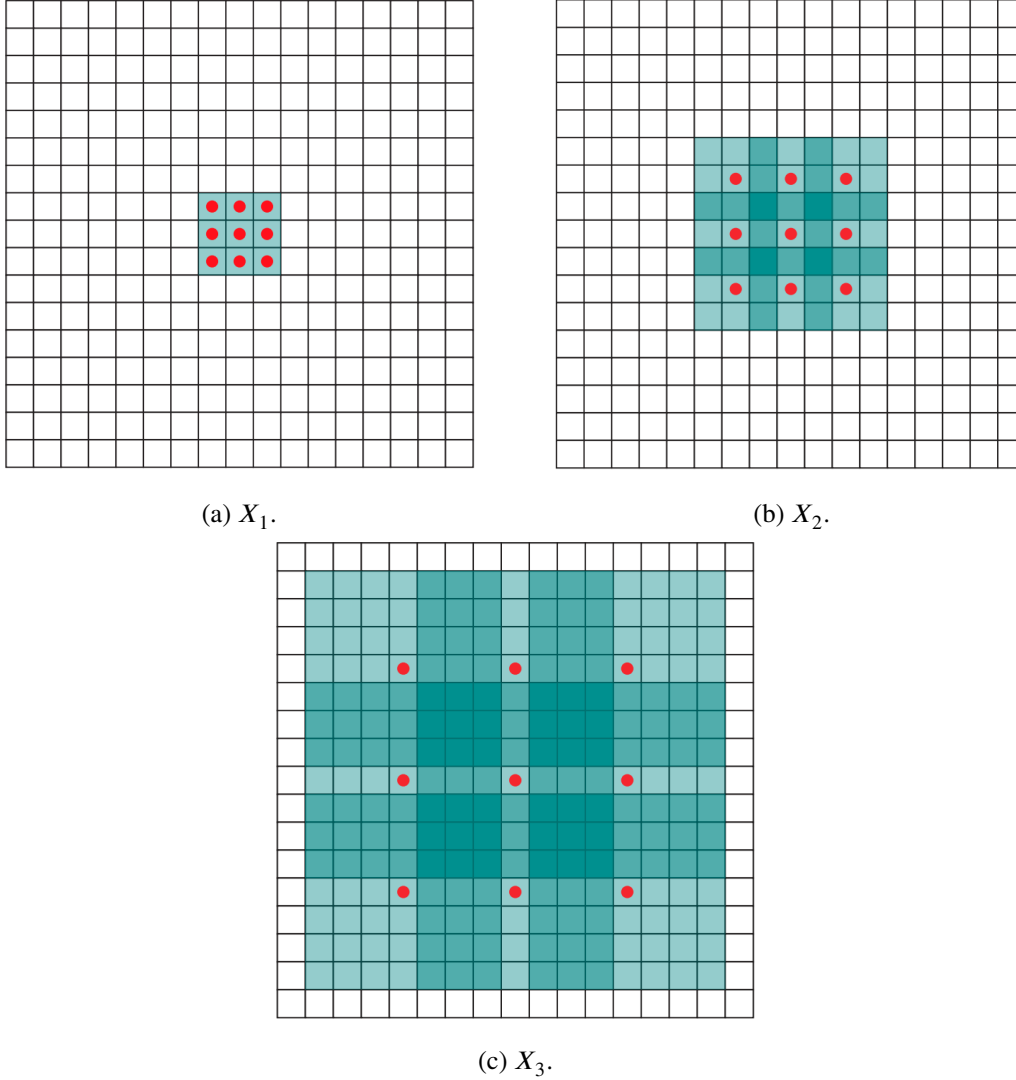


Figure 5.7: Atrous convolution supports exponential expansion of the receptive field without loss of resolution or coverage. (a) x_1 is produced from input by a 1-dilated convolution; each element in X_1 has a receptive field of 3×3 . (b) X_2 is produced from X_1 by a 2-dilated convolution; each element in X_2 has a receptive field of 7×7 . (c) X_3 is produced from X_2 by a 4-dilated convolution; each element in X_3 has a receptive field of 15×15 . The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly. The images are from [YK16].

pre-defined depth are calculated, the rate parameter is used to filter out edges. For any rate r only the edges with depths of the form $1 + r * i, i \in \mathbb{W}$ are added to the edge matrix E . Figure 5.8 shows an example of atrous convolution for graph with rate $r = 2$ and max depth of three.

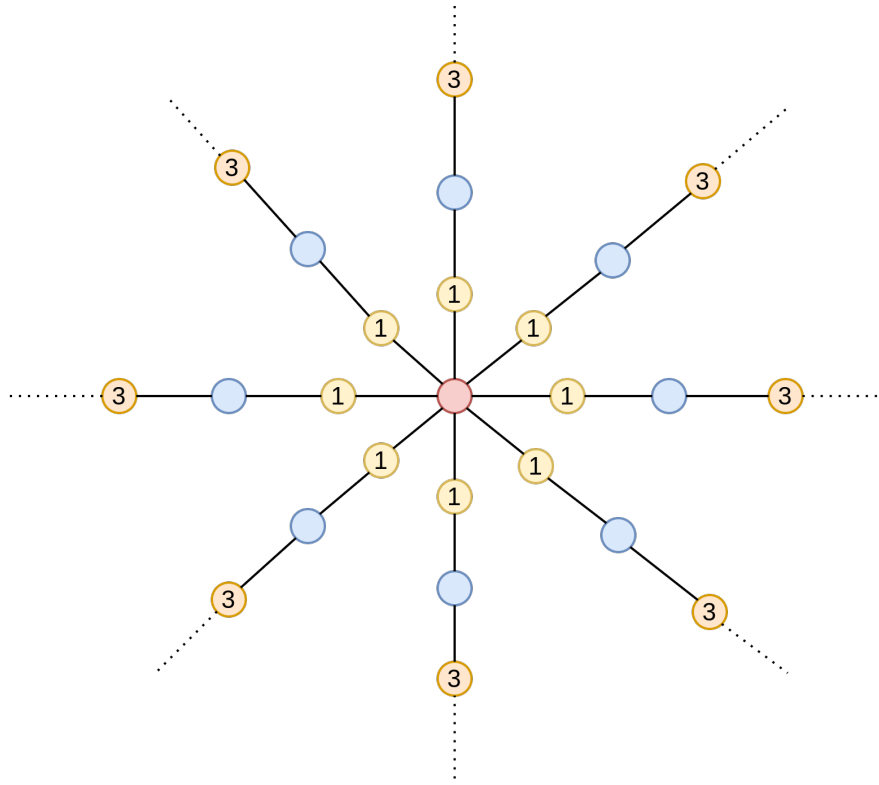


Figure 5.8: An example of atrous convolution on graph input. For the root vertex (in red) new edges with the vertices at depth one (in yellow) and depth three (in orange) are added to existing edges are added before applying any Graph Convolution.

Similar to atrous convolutions in images, atrous convolutions in graphs helps increase the receptive field of the model and helps the model to converge faster.

5.5.1 Depth Encoding

In atrous convolution for images due to the structured representation of images, the convolution operator has the information of mapping between input value and pixel position, but it is not the same in atrous convolution for graphs. Due to the permutation invariance in

the graph convolution operations, the model does not have any information about the depth of the neighbor. Hence in our work, to include the depth information, the Depth Encoding discussed in section 5.4 is extended to atrous convolutions.

The depth encoding function (equation 5.8) uses the calculated depths of the newly added edges to calculate their corresponding depth encodings. These depth encodings are added to the input features only once at the very beginning of the model. Adding depth encoding to atrous convolution helps the model to differentiate vertices based on both geodesic distance (distance along the graph) and on euclidean distance (distance in space). Using this information the model can decide the influence of a certain vertex on the final convolution.

6 Experimental Setup

6.1 Environment

For our work, the GPU cluster provided by the Institute for Robotics and Mechatronics of the DLR (German Aerospace Center), is used. The cluster consists of Titan V 12GB, Titan RTX 24GB, 2080 RTX Ti 11GB and Quadro GV100 32GB GPU's. Most of our training and testing is carried out on a single Titan V 12GB GPU.

Our code for the work is implemented using Python 3.8 and PyTorch 1.7 [Pas+19] is used for all Deep Learning related GPU operations. PyTorch-geometric [FL19] is a geometric deep learning extension library built for PyTorch. It provides data structures and functions to support graph data on a GPU. It also provides out of the box support for benchmark datasets and other Convolutional Graph Neural Network. In our work PyTorch-geometric is used for all graph-related operations. PyTorch lightning [Fal19] is a lightweight PyTorch wrapper for high-performance Machine Learning research. In our code PyTorch-Lightning is used to wrap training methods and for distributed training on multiple GPU's.

6.2 Datasets

Datasets are important in any learning-based methods. With the rise in the application of Deep Learning in almost every field, there has been quite a steep increase in the number of public datasets. There has also been a rise in the number of datasets targeting 3D scenes and objects, but most of these datasets are intended for the purpose of the classification rather than segmentation. Also due to the ease of obtaining point clouds over meshes, most of the datasets provide point clouds and not meshes.

Table 6.1 shows a few of the publicly available datasets for 3D segmentation, the format of data (point clouds or meshes) that is provided by the datasets, and the category (objects for 3D objects or scenes for 3D scenes). For our work, the datasets Coseg [Wan+12] consisting of high-resolution 3D meshes and ShapeNet [Cha+15] consisting of point clouds

are used for the training and evaluation. The datasets are chosen to show the performance of our work in this thesis on both point clouds and on meshes. Both the datasets are discussed in detail in the following sections.

Table 6.1: Shows few of the publicly available datasets for the task of 3D segmentation.

The first column is the name of the dataset, followed by the data format provided by the dataset, followed by the high level category of the 3D data provided by the dataset, followed by references to each of the datasets.

Dataset	Format	Category	Reference
COSEG	Meshes	Objects	[Wan+12]
LabeledPSB	Meshes	Objects	[KHS10] [CGF09]
PartNet	Meshes & Point Clouds	Objects	[Cha+15] [Mo+19]
ScanNet	Meshes & Point Clouds	Scenes	[Dai+17]
S3DIS	Point Clouds	Scenes	[Arm+16]
ShapeNet	Point Clouds	Objects	[Cha+15] [Yi+16]

6.2.1 COSEG

The Shape COSEG dataset [Wan+12] is a dataset consisting of high-resolution meshes of 3D objects and their part annotations. The dataset was introduced to provide data for quantitative analysis of how people consistently segment a set of shapes.

The labeling of the 3D models in the COSEG dataset is generated by using a semi-supervised learning method where the user actively assists in the co-analysis (simultaneous segmentation of the shapes in a set in a consistent matter) by iteratively providing inputs that progressively constraint the system. The method uses a novel constrained clustering technique which embeds elements to better respect their inter-distances in feature space together with the user-given set of constraints. Also an active learning method is introduced that suggests to the user where his input is likely to be the most effective in refining the results. Figure 6.1 shows an overview of the proposed method to obtain consistent labeling of the meshes.

The Shape COSEG dataset is a collection of eleven sets of shapes with a consistent ground-truth segmentation and labeling. Each set has a number of labels (or parts) ranging from three to five. The number of models in the first seven sets is quite small ranging

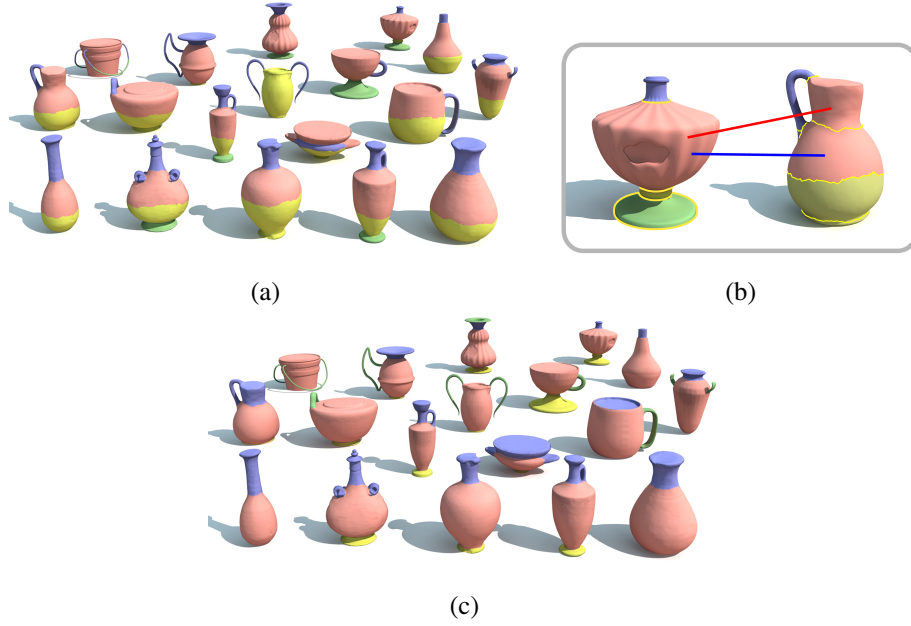
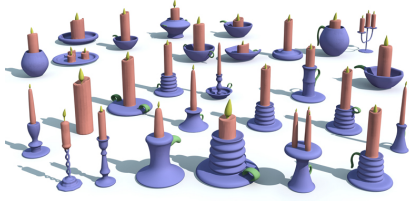


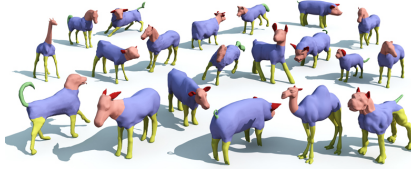
Figure 6.1: Overview of a single step in the active co-analysis from [Wan+12]: (a) Start with an initial unsupervised co-segmentation of the input set. (b) During active learning, the system automatically suggests constraints which would refine results and the user interactively adds constraints as appropriate. In this example, the user adds a cannot-link constraint (in red) and a must-link constraint (in blue) between segments. (c) The constraints are propagated to the set and the co-segmentation is refined. The process from (b) to (c) is repeated until the desired result is obtained.

from 12 44. For the last three sets (Tele-Aliens, Vases Large and Chairs Large) the number of models are slightly higher with 200, 300 and 400 each respectively. Figure 6.2 shows sample data from the COSEG dataset.

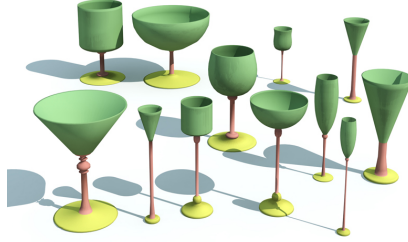
Table 6.2 shows the list of categories provided by the Shape COSEG along with the total number of 3D models in categories and the number of models used for training and during testing. Table 6.3 shows the number of labels in each of the categories in the COSEG dataset.



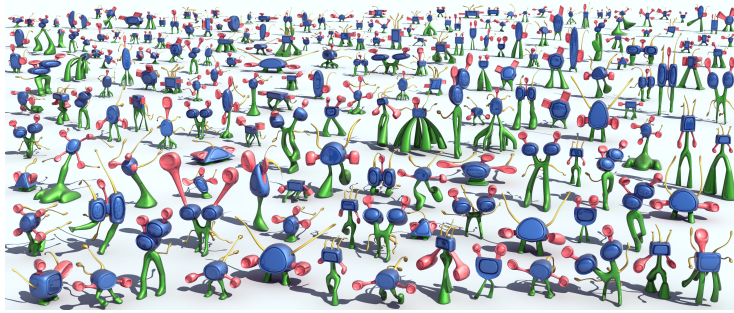
(a) Candelabra.



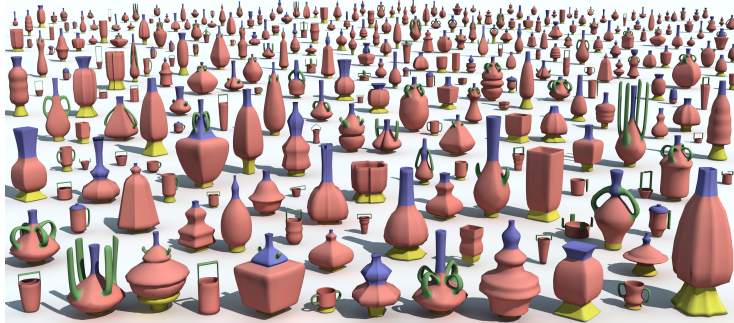
(b) Fourlegs.



(c) Goblets.



(d) Tele Aliens.



(e) Vases Large.

Figure 6.2: Sample meshes and their labelling from the Shape COSEG (COSEG) dataset. The images are from [Wan+12].

Table 6.2: COSEG: Starting from the first column shows the name of the categories, number of 3D models for that category, number of 3D models used for training and for testing.

The Shape COSEG dataset categories			
Category	# of models	Test	Train
Candelabra	28	12	16
Chairs	20	12	8
Fourleg	20	12	8
Goblets	12	6	8
Guitars	44	12	32
Lamps	20	12	8
Vases	28	12	16
Irons	18	12	6
Tele-aliens	200	12	188
Vases Large	300	12	288
Chairs Large	400	12	388

6.2.2 ShapeNet

ShapeNet part-annotation [Yi+16] is a dataset consisting of point clouds of various types of 3D objects collected from the 3D objects in ShapetNetCore [Cha+15] and their part annotations. ShapeNet is an ongoing effort to establish a richly-annotated, large-scale dataset of 3D shapes. It provides researchers around the world with this data to enable research in computer graphics, computer vision, robotics, and other related disciplines. ShapeNet is a collaborative effort between researchers at Princeton, Stanford, and TTIC. ShapeNet is organized according to the WordNet hierarchy. ShapeNetCore is a subset of the full ShapeNet dataset with single clean 3D models and manually verified category and alignment annotations. It covers 55 common object categories with about 51,300 unique 3D models. ShapeNet part-annotation dataset provides part annotations (annotation for each of differentiable parts of an object eg. legs, hands, face, etc. in a human) for a subset of models from ShapeNetCore.

The data for the ShapeNet part-annotation dataset is generated using a novel active learning

Table 6.3: COSEG: Shows for each category in Shape COSEG dataset, the number of labels.

Shape COSEG dataset labels	
Category	# of labels
Candelabra	4
Chairs	3
Fourleg	5
Goblets	3
Guitars	3
Lamps	3
Vases	4
Irons	3
Tele-aliens	4
Vases Large	4
Chairs Large	3
Total	39

method that is capable of enriching massive geometric datasets with accurate semantic region annotations. Similar to COSEG dataset, the proposed method involves cycling between manually annotating the regions, automatically propagating these annotations across the rest of the shapes, manually verifying both human and automatic regions and learning from the verification results to improve the automatic propagation algorithms. The automatic propagation of the human labels across a dynamic shape network is done using a Conditional Random Field (CRF) framework, that takes advantage of global shape-to-shape similarities, local feature similarities, and point-to-point correspondences. They also include a utility function that explicitly models the time cost of human input across all steps of the method to jointly optimize for the set of models to annotate and for the set of models to verify based on the predicted impact of these actions on human efficiency.

The ShapeNet part-annotation dataset consists of over 16000 point clouds of objects in 16 categories with the number of models in each category varying from around 50 to around 3000. An example point cloud for each category can be seen in figure 6.4. Table 6.4 shows the names of the categories and number of models provided by the dataset in each category and the number of 3D models used for training, validation, and testing the algorithms. Each

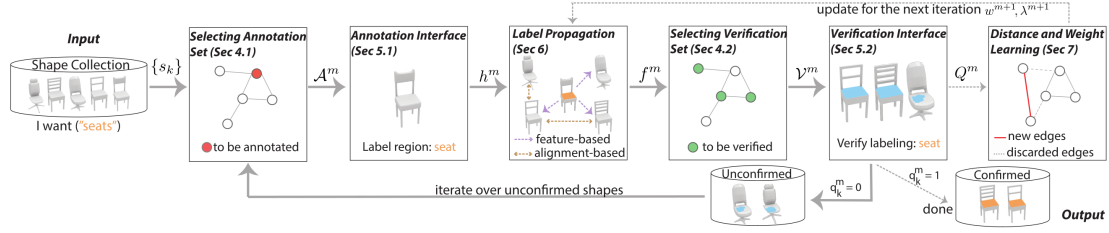


Figure 6.3: Overview of the ShapeNet part-annotation pipeline used to generate the labeling of 3D models from [Yi+16]. Given the input dataset, an annotation set is selected and UI is used to obtain human labels. The method automatically propagate these labels to the rest of the shapes and then query the users to verify the most confident propagations. Then these verifications are used to improve our propagation technique.

category is split into multiple meaningful part labels, with the number of parts ranging from two to six. In total there are 50 labels over 16 categories. Table 6.5 shows the number of labels for each of the category and their names for a better understanding of the labels/parts.

Since point clouds have no connectivity information to form a graph, edges are created for a vertex with its nearest neighbors (in euclidean space) in the point cloud. To do this a KNN algorithm with $k = 8$ is applied on the point cloud data to transform it into a graph where the edges for a vertex are its k nearest neighbors.

6.3 Models

For any Deep Learning task model architecture has a huge impact on the performance. There is a lot of research happening in finding new model architectures or modifying existing architectures to try increasing the model performance. In our work, the focus is not on improving a model architecture, but on using existing model architectures and incorporate our work and other related works into these. This is to ensure a consistent comparison of the performance of our work with other related Graph Convolutions. In our work, a total of three different model architectures are used for the task of 3D segmentation. The first model is the transformer model discussed in section 4.3, the second model architecture is inspired by FeaStNet architecture [VBV18] and the last is a U-Net model with Graph Convolutions. Each of the models are discussed in detail in the following sections.

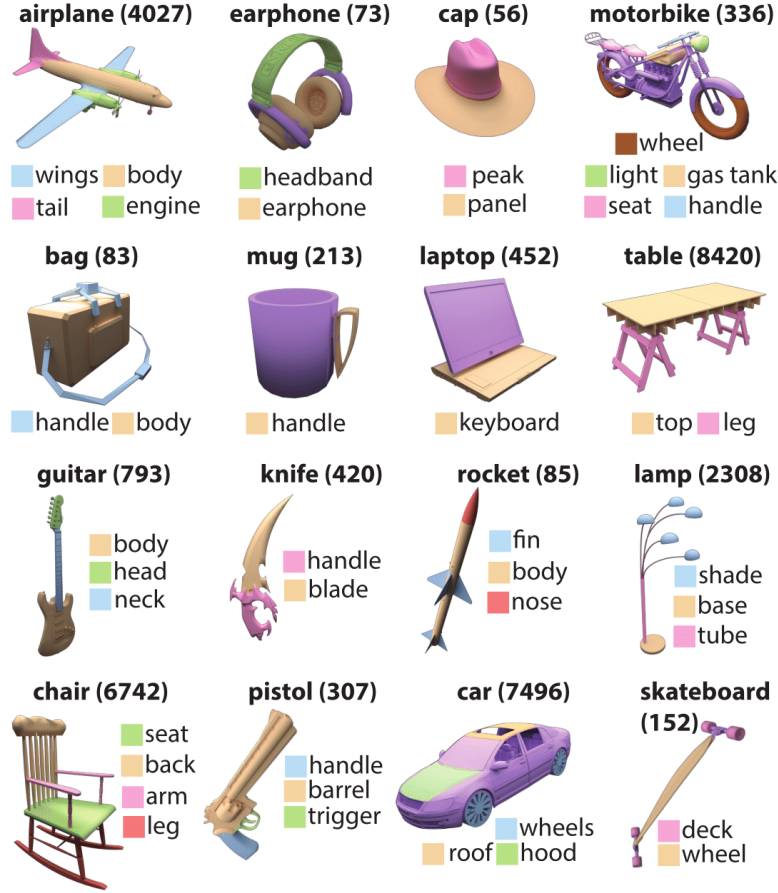


Figure 6.4: Shows an example of one point cloud for each category from the ShapeNet part-annotation dataset and their labels. The image is from [Yi+16].

6.3.1 Transformer

The first model used for the task of Mesh segmentation is the transformer network for meshes discussed in section 4.3. The model used in our work consists of **three** encoder layers and **three** decoder layers. Following are the sequence of steps involved in a single forward pass

- The input to the model is the concatenation of the positions of the vertices and the normals associated, thus forming a six-dimensional feature vector for each vertex $x_i \in \mathbb{R}^6$.
- This batched input $X = \{x_1, \dots, x_N\}$ is first passed into two feed-forward neural

Table 6.4: ShapeNet: Starting from the first column shows the name of the categories, number of 3D models for that category, number of 3D models used for training, used for validation and used for testing.

ShapeNet Part-annotation categories				
Category	# of models	Train	Validation	Test
Airplane	2690	1958	391	341
Bag	76	54	8	14
Cap	55	39	5	11
Car	898	659	81	158
Chair	3758	2658	396	704
Earphone	69	49	6	14
Guitar	787	550	78	159
Knife	392	277	35	80
Lamp	1547	1118	143	286
Laptop	451	324	44	83
Motorbike	202	125	26	51
Mug	184	130	16	38
Pistol	283	209	30	44
Rocket	66	46	8	12
Skateboard	152	106	15	31
Table	5271	3835	588	848
Total	16881	12137	1870	2874

networks to increase the feature dimensions to 64 and then to 256 (also referred to depth of the model d_{model}) to generate a 256 dimensional vectors $h_i \in \mathbb{R}^{256}$ for each vertex.

- Using the edges E of the graph, a neighbor tensor $H' \in \mathbb{R}^{N \times m' \times 256}$ (m' is the maximum degree of all the vertices, N is the number of vertices) is generated where each row i consists of the vertex i features h_i and features of all the neighboring vertices to the vertex \mathcal{N}_{h_j} . Not all the vertices have the same degree, hence to form a dense matrix a padding of zero values is added to the rows of vertices for which

Table 6.5: Shows the number of labels for each category and their names for better understanding. Unnamed label refers to a label that exists in the annotations but does not have a name that can describe it.

ShapeNet Part-annotation categories and labels		
Category	# of Labels	Label Names
Airplane	4	Wings, Body, Tail, Engine
Bag	2	Handle, Body
Cap	2	Peak, Panel
Car	4	Body, Wheels, Roof, Hood
Chair	4	Seat, Back, Arm, Leg
Earphone	3	Headband, Earphone, Connector
Guitar	3	Body, Head, Neck
Knife	2	Handle, Blade
Lamp	4	Shade, Base, Tube, Unnamed
Laptop	2	Screen, Keyboard
Motorbike	6	Body, Wheel, Light, Gas Tank, Seat, Handle
Mug	2	Mug, Handle
Pistol	3	Handle, Barrel, Trigger
Rocket	3	Fin, Body, Nose
Skateboard	3	Deck, Wheel, Axle
Table	3	Top, Leg, Unnamed
Total	50	-

the degree is less than the max degree m' .

- The neighbor matrix H' is passed through the encoder stacks. Each of the multi-head attention in the encoder stacks consists of eight attention heads. Inside the point-wise feed-forward neural network, the depth is increased to $d_f = 1024$ and then back to 256.
- The output tensor of the encoder stack $Z \in \mathbb{R}^{n \times m' \times 256}$ along with the input features X is passed to the decoder stack.
- The output of the decoder stack is passed through a feed-forward neural network

with the number of output features the same as the number of output classes (50 for ShapeNet and 39 for COSEG).

6.3.2 FeaStNet

The second model architecture is inspired by the FeaStNet [VBV18] architecture which was originally introduced to demonstrate the performance of Feature-Steered Graph Convolutions for 3D part labelling.

The model consists of a sequence of feed-forward neural networks and graph convolutions as shown in the figure 6.5. The network also consists of max pooling and global max-pooling layers. Max-pooling refers to max-pooling of neighboring node features where the output is the maximum value from the root vertex and its neighbors. Global max pooling refers to max-pooling of all the vertices in the graph. It outputs the maximum value from all the vertices of the graph. Each of the layers (linear, graph convolutions, pooling) are followed by a LeakyRelu (slope=0.1) activation layers.

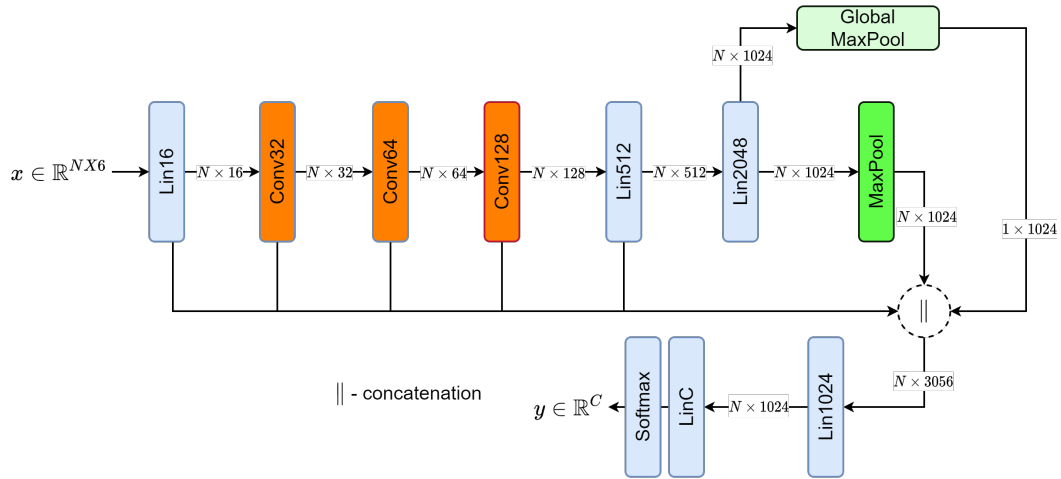


Figure 6.5: Model Architecture of the FeaStNet. $\text{Lin}P$ corresponds to a simple feed-forward neural network with P output channels. $\text{Conv}P$ corresponds to a graph convolution with P output channels. MaxPool is pooling operation over features of each vertex. Global MaxPool is max pooling operation over features of all the vertices. Each of the sublayers are followed by a LeakyRelu (slope=0.1) activation layers.

Following are the sequence of steps involved in a single forward pass in FeaStnet model

- Similar to the transformer network the input is a six-dimensional vector $x_i \in \mathbb{R}^6$ which is a concatenation of the position and the normal of a vertex.
- The input is first passed through a single feed-forward neural network to increase the feature dimensions to 16.
- The output is then passed through three graph convolution layers with layer doubling the output feature dimensions, finally generating a 128 dimensional feature vectors for each vertex in the graph.
- Then two feed-forward neural networks are applied to increase the feature dimensions to 1024
- Then a MaxPool and global MaxPool operations are performed.
- All the outputs from the above layers are concatenated to form a 3056 dimensional vector which is passed through two feed-forward networks and a softmax layer to output the probabilities of each label.

6.3.3 U-Net

The third model used to test our approach is inspired by the U-Net architecture [RFB15] which was first developed for biomedical image segmentation. U-Net generate fast and precise segmentation of images. Since their introduction in 2015 U-Nets have been used for various Image-to-Image translation tasks such as normal generation, depth map generation and also in Image-to-Image adversarial networks such as Pix2Pix [Iso+18].

The architecture consists of two paths, a contracting path to capture context and a symmetric expanding path that enables precise localization. Figure 6.6 shows an example U-Net architecture for an image from [RFB15]. In our work a similar version is used where 2D convolution operations are replaced with Graph Convolutions, max-pooling and up-convolutions are replaced with edge-contraction pooling [Die19] and edge un-pooling operations discussed in detail in section 6.3.3.

Following are the sequence of steps involved in a single forward pass in graph U-Net model

- Similar to the other models the input is a six-dimensional vector $x_i \in \mathbb{R}^6$ which is a concatenation of the position and the normal of a vertex.

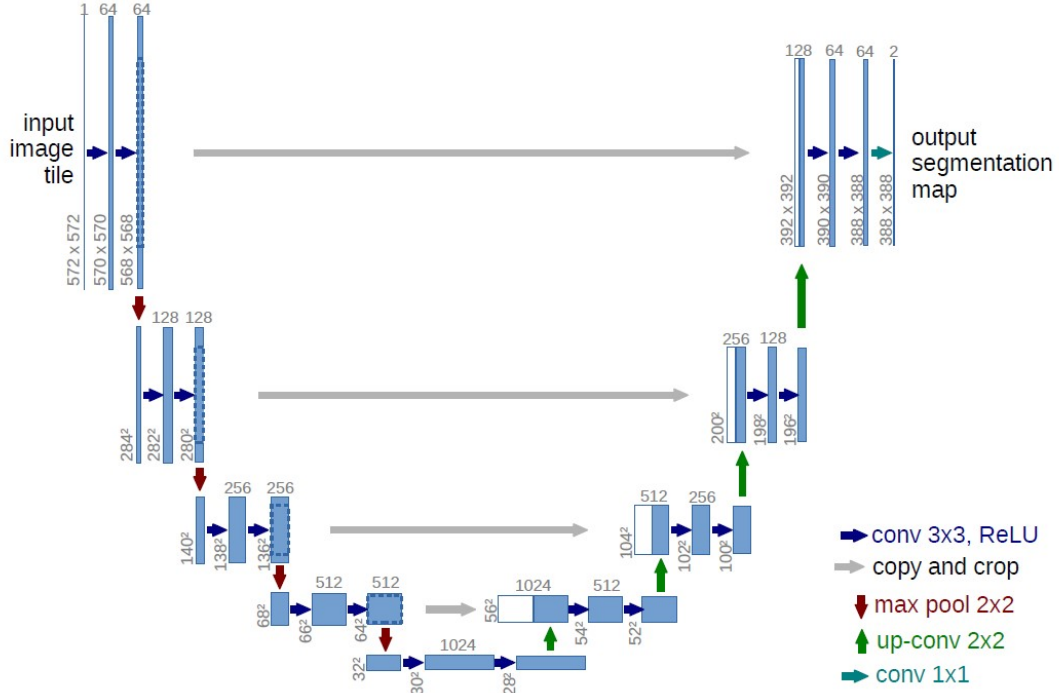


Figure 6.6: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. The image is from [RFB15].

- The input is first passed through a single feed-forward neural network to increase the feature dimensions to 16.
- The output is then passed through three steps (first path) of one graph convolution layer and edge pooling layer each, with each step doubling the output feature dimensions and halving the number of vertices in the graph.
- Then the output is passed through three steps (second path) of one graph convolution layer and edge un-pooling layer each. After each step the output feature dimensions are halved and the number of vertices in the graph are doubled. Also at each step a skip connection from the output of the first path is added as shown in the figure 6.7.
- The output is then passed through two feed-forward networks and a softmax layer to

output the probabilities of each label.

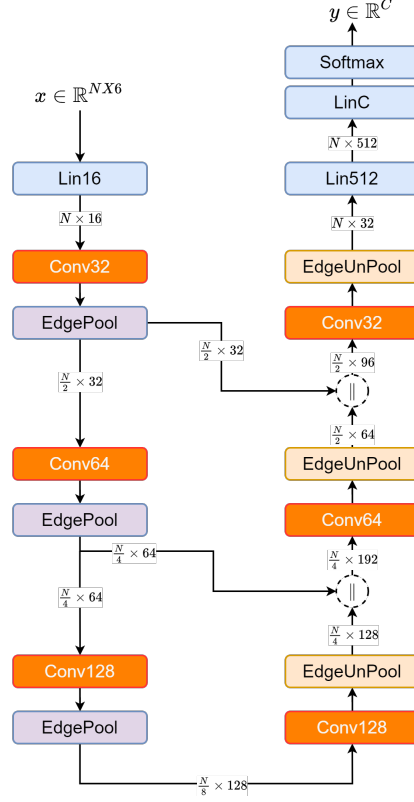


Figure 6.7: U-net architecture for graphs. $\text{Lin}P$ corresponds to a simple feed-forward neural network with P output channels. $\text{Conv}P$ corresponds to a graph convolution with P output channels. Edge pooling and unpooling are pooling operations explained in section 6.3.3.

Edge Pooling and Unpooling

Edge-Contraction Pooling [Die19] is a pooling operation for a graph, when applied results in a coarse representation of the input graph. Edge contraction means merging of two nodes $e = \{v_i, v_j\}$ to create a new node v_e and new edges such that the new node v_e is adjacent to all nodes v_i or v_j have been adjacent to. In the newly formed graph v_i , v_j and all their edges are deleted. Such a contraction reduces the number of nodes in a graph by one.

Edge contraction pooling defines a methodology to choose which set of edges should be contracted and the node features should be combined. Edge scoring functions are used to

decide the set of edges to be contracted. First a raw edge score r is computed for each of the edges using $r(e_{ij}) = W^T(x_i \parallel x_j) + b$, where $W \in \mathbb{R}^F, b \in \mathbb{R}$ are learnable parameters, $x_i, x_j \in \mathbb{R}^F$ are node features and \parallel is a concatenation operation.

Using the raw edge scores the actual node scores are computed. Two construction methods are proposed *tanh*: $s_{ij} = \tanh(r_{ij})$ and *softmax*: $s_{ij} = \text{softmax}_{r_{*j}}(r_{ij})$ where the softmax is applied over all edges which end in the same node. In our U-Net softmax-based edge contraction is used. Once the edge scores are calculated all the edges are sorted by their score and successively choose the edge with the highest score whose two nodes have not yet been part of a contracted edge.

After contracting the new node features are formed by adding the features of the nodes of the contracted edge and weighting them with the corresponding edge score $x'_{ij} = s_{ij}(x_i + x_j)$

During the un-pooling, the pooling information is used to check which edges have been contracted and the contraction is un-done. After un-doing the contraction the contracted node features are copied to the now un-contracted nodes.

6.4 Training

During the training all the three models are trained for 1000 epochs with a batch size of 4. Adam [KB17] optimizer with beta1 0.9 and beta2 0.99 is used with a learning rate of 6e-5, 1e-4, 1e-4 for the transformer network, FeaStNet and U-Net respectively.

6.4.1 Loss

The loss between the predicted label probabilities y and the target probabilities y^t is a cross-entropy loss defined by

$$loss = - \sum_i^C y_i^t \log(y_i) \quad (6.1)$$

where C is the total number of target labels.

6.5 Evaluation

During the evaluation, the epoch with the best validation results is used to load the model parameters and report the results. Mean Intersection-Over-Union (mIoU) metric is used to measure the performance of the algorithms.

Intersection-Over-Union (IoU) is an evaluation metric typically used to measure overlap between two bounding boxes or masks. For images and bounding boxes/masks IoU is calculated as follows

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (6.2)$$

IoU for graph is calculated for each label (l_i) in a sample as follows

$$\text{IoU}_{l_i} = \frac{TP_{l_i}}{TP_{l_i} + FP_{l_i} + TN_{l_i}} \quad (6.3)$$

where TP_{l_i} , FP_{l_i} and TN_{l_i} are the number of true positives, false positives and true negatives for the label l_i respectively.

mIoU is calculated using the standard experimental protocol for graph segmentation from [VBV18] [Qi+17a], involving the following steps

- First for each sample, predicted labels for each of the vertices are calculated using the output probabilities. The predicted label for a vertex is the label associated with the maximum probability among the probabilities of the labels from the category of the sample. For example, for a sample from the car category only the probabilities of the labels from the car category (Body, Wheels, Roof, Hood) are used to find the predicted label.
- Then using the predicted labels and target labels IoU is calculated for each of the labels in all the categories.
- Then mIoU for each category is the mean of the IoUs of all the labels in that category.

$$\text{mIoU}_{c_i} = \frac{\sum_i^{L_{c_i}} \text{IoU}_{l_i}}{L_{c_i}} \quad (6.4)$$

where L_{c_i} is the number of labels in c_i category.

- Average mIoU is the mean of the mIoUs of all categories in the dataset.

$$\text{Average mIoU} = \frac{\sum_i^C \text{mIoU}_{c_i}}{C} \quad (6.5)$$

- Average instance mIoU is the mean of the IoUs of all labels in the dataset.

$$\text{Average Instance mIoU} = \frac{\sum_i^L \text{IoU}_{l_i}}{L} \quad (6.6)$$

where L is the total number of labels across all the categories in the dataset.

6.6 Implementation

6.6.1 Batching

Batching in graph data works very differently compared to batching in images. A batch size of four in images implies four images with a resolution of $H \times W$. The same can not be said for a graph as not all the graphs have the same number of vertices and edges. In graph data batching is done by combining multiple graphs into a single graph with updated edge indices (updated with offsets) and vertices. Hence in graphs, a batch size of four is internally transformed into a single graph $G_{batch} : (V_{batch}, E_{batch})$ with number of vertices $|V_{batch}| = \sum_{i=1}^4 |V_i|$ and number of edges $|E_{batch}| = \sum_{i=1}^4 |E_i|$.

6.6.2 Breadth First Search

In section 5.5 atrous convolution for graphs was introduced. To apply atrous convolution new edges with depth less than a fixed depth should be added to the existing edges. To generate this data a Breadth First Search (BFS) is performed on the graph for each vertex. Breadth First Search is an algorithm for traversing or searching tree or graph data structures and is the most used method to find distances from a root vertex to all the vertices. For each vertex, to produce its new edges one BFS traversal starting from the root vertex is required. Hence for a graph to produce all the new edges N (number of vertices) BFS traversals are required which is an expensive operation.

For model architectures such as MeshTrans, FeaStNet where the graph structure is retained through the entire forward pass, the new edges can be pre-computed and can be stored as part of training data. But for architectures such as U-Net where the graph structure is not retained, these new edges should be calculated dynamically every time the graph structure changes. To perform the BFS traversals on a CPU would take minutes for each batch hence increasing the training time significantly. Hence a GPU implementation of the BFS traversal that can be parallelized is required.

For our thesis, a BFS extension to PyTorch is implemented in CUDA. The algorithm is inspired from the paper [MGG12] which presents a BFS parallelization that achieves an asymptotically optimal $O(|V| + |E|)$ work complexity.

7 Results

In this chapter, the results of our work in this thesis are presented and are compared to other related works. First, the results from the COSEG dataset are shown and then, the results from the ShapeNet dataset are shown.

7.1 Model configurations

As discussed in section 6.3 a total of three model architectures namely

- FeaStNet [VBV18]
- U-Net [RFB15]
- Transformer for Mesh Segmentation (MeshTrans) (Ours)

are used in our experimental setup. Since the models FeaStNet and U-Net have an underlying graph convolution layer that can be configured, these models are permuted with three different Graph Convolution layers namely

- Transformer Convolution (TransConv) (Ours)
- Feature Steered Graph Convolution (FeaSt GraphConv) [VBV18]
- Graph Attention Network (GAT) [Vel+18]

as shown in the table 7.1, where for each of the model we compare our graph convolution layer TransConv with the other two graph convolution layers. For our model MeshTrans, since there is no underlying graph convolution layer that can be configured, it is considered as a single model configuration.

Results for each of the seven configurations of model architectures as shown in the table 7.1 on the two datasets (The Shape COSEG and ShapeNet) are shown in this chapter.

Table 7.1: Shows different configurations of models and graph convolution layers that are tested in our thesis. The rows are the three different models discussed in section 6.3 and the columns are different graph convolution layers used inside the model architecture. For MeshTrans, there are no graph convolution layers that can be configured.

Model \ Convolution	TransConv (Ours)	FeaSt GraphConv	GAT
FeaStNet	FeaStNet + TransConv	FeaStNet + FeaSt	FeaStNet + GAT
U-Net	U-Net + TransConv	U-Net + FeaSt	U-Net + GAT
MeshTrans (Ours)	-	-	-

7.2 The Shape COSEG

In this section, the mIoUs for each of the eleven categories along with the average mIoUs for the Shape COSEG dataset is shown.

Table 7.2 and figure 7.1 shows the per-category mIoUs for the models FeaStNet + FeaSt, FeaStNet + GAT and our methods FeaStNet + TransConv and MeshTrans. It can be seen that our graph convolution layer TransConv outperforms all the other models in most of the categories. However, our approach to use the transformer networks directly for mesh segmentation (MeshTrans) does not produce any state of the art results, but has a performance comparable to the other methods. This performance could be attributed to the lack of message propagation between neighbors in a MeshTrans. Unlike other Convolutional Graph Neural Network (ConvGNN), due to the design of our MeshTrans the neighbor node representations are never updated with the information from their neighboring nodes. Since all the meshes are of high resolution, all the neighbors are close to each other thus without message propagation, the model can not make accurate predictions.

Table 7.3 and shows the per-category mIoUs for all the eleven categories for the models U-Net + FeaSt, U-Net + GAT and our work U-Net + TransConv. MeshTrans is not compared in this table because the number of model parameters in MeshTrans is significantly higher compared to U-Net models hence will not be a reasonable comparison.

It can be seen that, similar to the results on FeaStNet base model, our graph convolution layer TransConv using U-Net base model outperforms all the other models in most of the categories and has a comparable performance in the rest of the categories. Similar to results

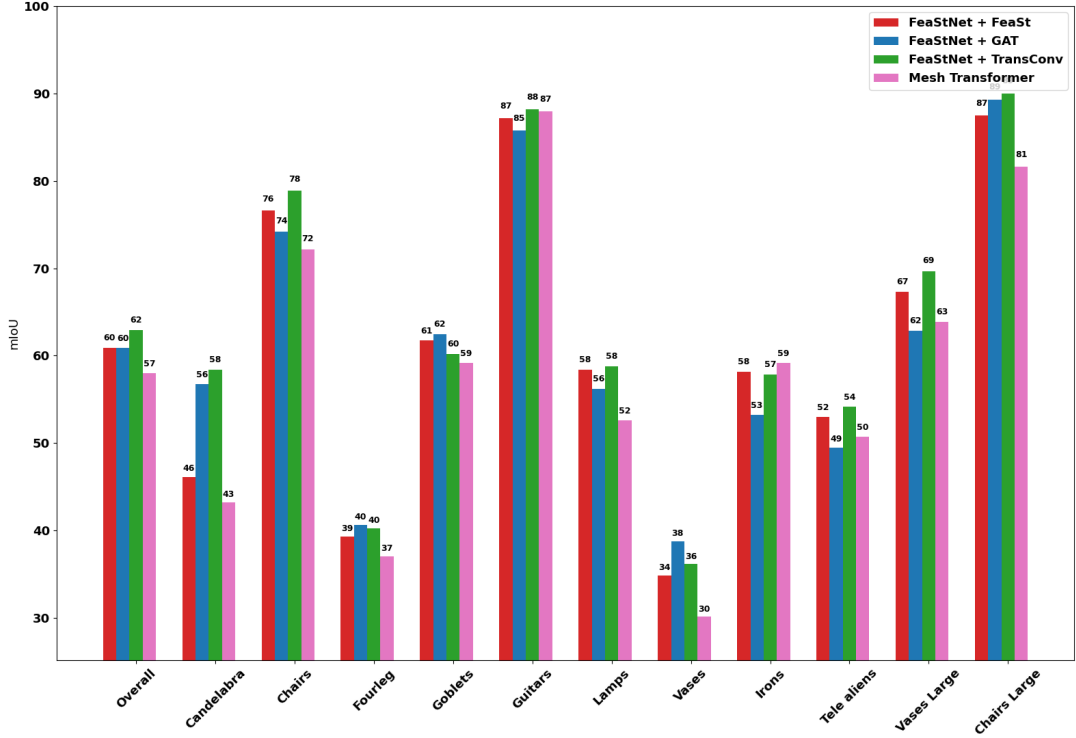


Figure 7.1: Segmentation accuracy in mIoU on the COSEG dataset for FeaStNet + FeaSt, FeaStNet + GAT, FeaStNet + TransConv, MeshTrans. mIoU values are shown on the y-axis and the categories are on the x-axis. Each data point shape represents a different model. The green and the pink bar represents our works TransConv and MeshTrans respectively.

Table 7.2: Segmentation accuracy in mIoU on the COSEG dataset for FeaStNet + FeaSt, FeaStNet + GAT, FeaStNet + TransConv, MeshTrans.

	Overall	Candelabra	Chairs	Fourleg	Goblets	Guitars	Lamps	Vases	Irons	Tele aliens	Vases Large	Chairs Large
FeaStNet + FeaSt	60.91	46.09	76.6	39.33	61.72	87.19	58.41	34.87	58.17	52.99	67.28	87.45
FeaStNet + GAT	60.87	56.72	74.16	40.66	62.46	85.76	56.24	38.75	53.25	49.45	62.88	89.30
FeaStNet + TransConv (Ours)	62.96	58.43	78.87	40.25	60.18	88.18	58.81	36.16	57.87	54.19	69.64	89.99
MeshTrans (Ours)	57.97	43.19	72.16	37.06	59.17	87.96	52.61	30.16	59.16	50.74	63.85	81.63

with FeaStNet models, our approach to use transformers directly for mesh segmentation (MeshTrans) has a relatively poor performance.

Table 7.3: Segmentation accuracy in mIoU on the COSEG dataset for U-Net + FeaSt, U-Net + GAT and U-Net + TransConv (ours) model configurations.

	Overall	Candelabra	Chairs	Fourleg	Goblets	Guitars	Lamps	Vases	Irons	Tele aliens	Vases Large	Chairs Large
U-Net + FeaSt	62.74	52.03	72.16	42.37	70.81	83.36	53.56	36.89	66.51	60.96	68.26	83.3
U-Net + GAT	62.13	50.9	70.96	42.98	65.84	83.89	55.2	38.82	66.33	62.19	62.97	83.44
U-Net + TransConv (Ours)	62.94	52.85	73.18	42.18	71.51	82.83	56.77	35.76	67.19	62.44	64.58	83.12

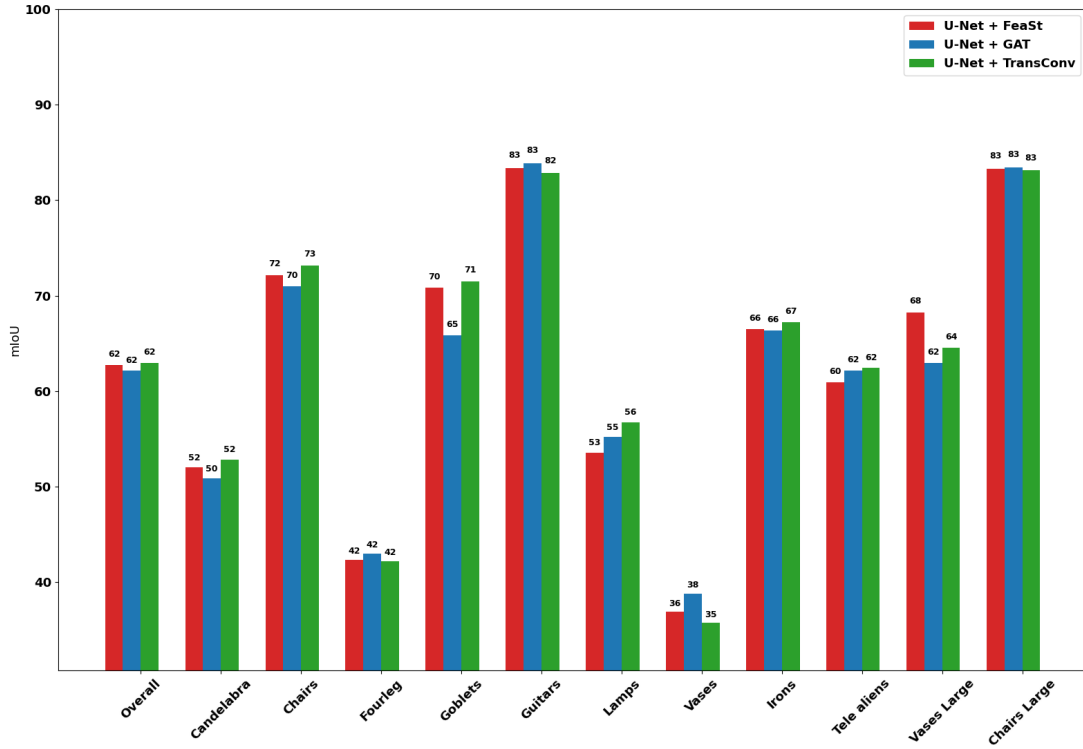


Figure 7.2: Segmentation accuracy in mIoU on the COSEG dataset for U-Net + FeaSt, U-Net + GAT, U-Net + TransConv. mIoU values are shown on the y-axis and the categories are on the x-axis. Each data point shape represents a different model. Green bar shows results for our graph convolution layer TransConv.

7.3 ShapeNet

In this section, the mIoUs for each of the 16 categories along with the average mIoUs for the ShapeNet part-annotation dataset is shown.

For each combination of the Graph Convolution layer in FeaStNet model table 7.4 and figure 7.3 shows the per-category mIoU results. It can be seen that our graph convolution layer (TransConv) outperforms both Graph Attention Network and FeaSt Graph Convolutions in most of the categories and has comparable performance in the rest. Also, although our Transformer for Mesh Segmentation (MeshTrans) did not outperform the other approaches in most of the categories, it has comparable performance to the others. The weaker performance could be attributed to the low receptive field in MeshTrans compared to the FeaStNet model. In Convolutional Graph Neural Network after each convolution, the nodes contain information about their neighbors, thus increasing the receptive field exponentially, whereas in MeshTrans the receptive field is constant (depending on the k in KNN algorithm) which is decided when providing the input to the network. However, our MeshTrans has significantly high performance in the Cap category compared to the other models, which shows the potential of our MeshTrans.

Table 7.4: Segmentation accuracy in mIoU on the ShapeNet dataset for FeaStNet + FeaSt, FeaStNet + GAT, FeaStNet + TransConv (Ours) and MeshTrans (Ours) model configurations.

	Overall	Airplane	Bag	Cap	Car	Chair	Earphone	Guitar	Knife	Lamp	Laptop	Motorbike	Mug	Pistol	Rocket	Skateboard	Table
FeaStNet + FeaSt	76.21	76.54	71.94	76.17	73.72	85.14	63.7	88.57	77.34	79.52	95.97	60.15	90.63	78.1	55.63	70.65	75.67
FeaStNet + GAT	76.74	77.01	70.82	80.28	71.48	83.47	61.39	88.93	77.48	78.24	95.73	62.88	91.39	77.83	62.27	73.49	75.12
FeaStNet + TransConv (Ours)	77.2	78.1	72.69	80.98	72.19	86.28	62.65	89.01	77.54	79.77	96.04	61.76	92.7	80.41	57.49	71.54	76.13
MeshTrans (Ours)	76.08	75.16	71.8	84.06	71.53	84.82	62.14	88.5	77.49	78.11	95.77	58.89	91.21	79.88	53.25	72.68	73.79

Table 7.4 and figure 7.4 shows mIoU results for the model U-Net + FeaSt, U-Net + GAT and our method U-Net + TransConv for each of the 16 categories in the ShapeNet datasets. Again our MeshTrans is not compared with U-Net because of the significantly different number of model parameters. As shown in the results, our graph convolution layer TransConv outperforms other graph convolution layers in most of the categories and has a comparable performance in the rest of the categories.

7.3.1 Depth Encoding

Table 7.6 and figures 7.5, 7.6, 7.7 shows results for our works with and without our depth encodings discussed in section 5.4 that are added to the input. It can be seen that in model architectures FeaStNet and U-Net the depth encoding helps improve the model performance.

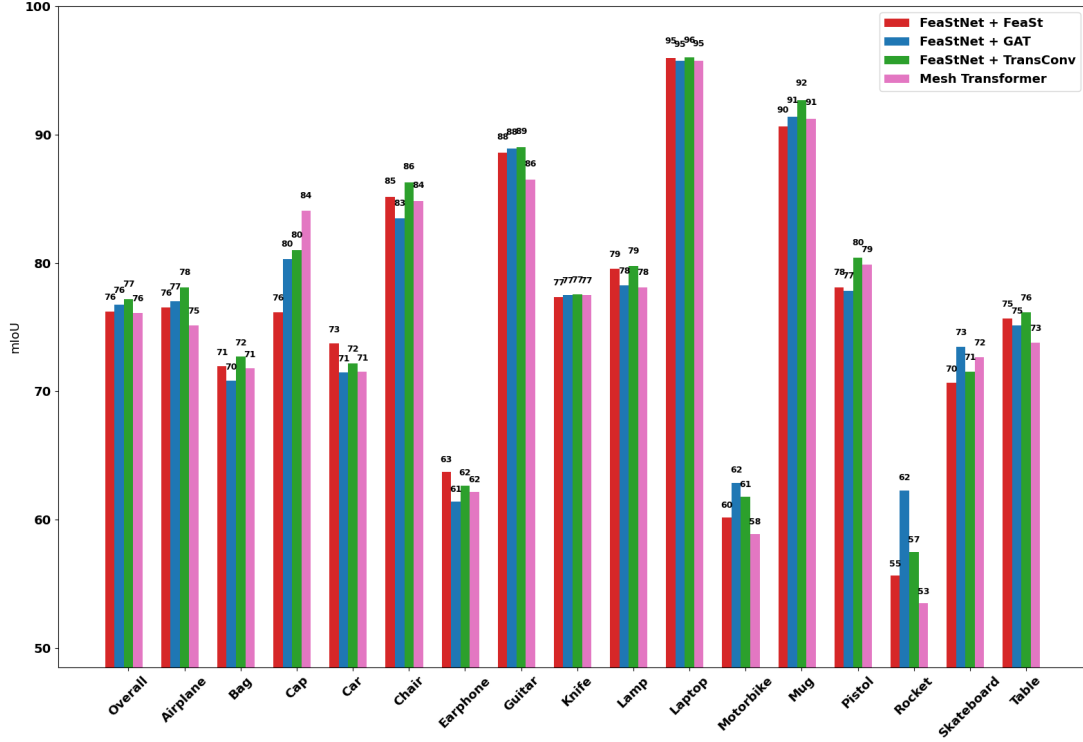


Figure 7.3: Segmentation accuracy in mIoU on the ShapeNet dataset for FeaStNet + FeaSt, FeaStNet + GAT, FeaStNet + TransConv, MeshTrans. mIoU values are shown on the y-axis and the categories are on the x-axis. Each bar represents a different model as shown in the plot legend. The green and the pink bar represents our works TransConv and MeshTrans respectively.

Table 7.5: Segmentation accuracy in mIoU on the ShapeNet dataset for U-Net + FeaSt, U-Net + GAT and U-Net + TransConv (ours) model configurations.

	Overall	Airplane	Bag	Cap	Car	Chair	Earphone	Guitar	Knife	Lamp	Laptop	Motorbike	Mug	Pistol	Rocket	Skateboard	Table
U-Net + FeaSt	71.49	73.66	70.5	71.23	61.85	79.84	63.84	87.12	76.71	73.1	94.31	49.41	91.32	69.97	53.01	61.61	66.5
U-Net + GAT	70.24	70.18	69.1	72.74	61.16	78.65	64.12	86.6	75.17	73.73	94.85	53.6	86.91	67.57	42.18	62.16	65.19
U-Net + TransConv (Ours)	71.31	71.67	76.45	69.76	63.19	80.19	64.27	87.17	77.05	72.8	94.32	48.91	85.71	70.01	43.01	62.56	68.91

However, in MeshTrans depth encoding does not improve the performance. The main reason for this is that in MeshTrans due to the conditioning on the decoder, the network already knows which vertex is the root vertex whereas in FeaStNet and U-Net without depth encoding, the model does not know which vertex is the root vertex, hence depth encoding

7 Results

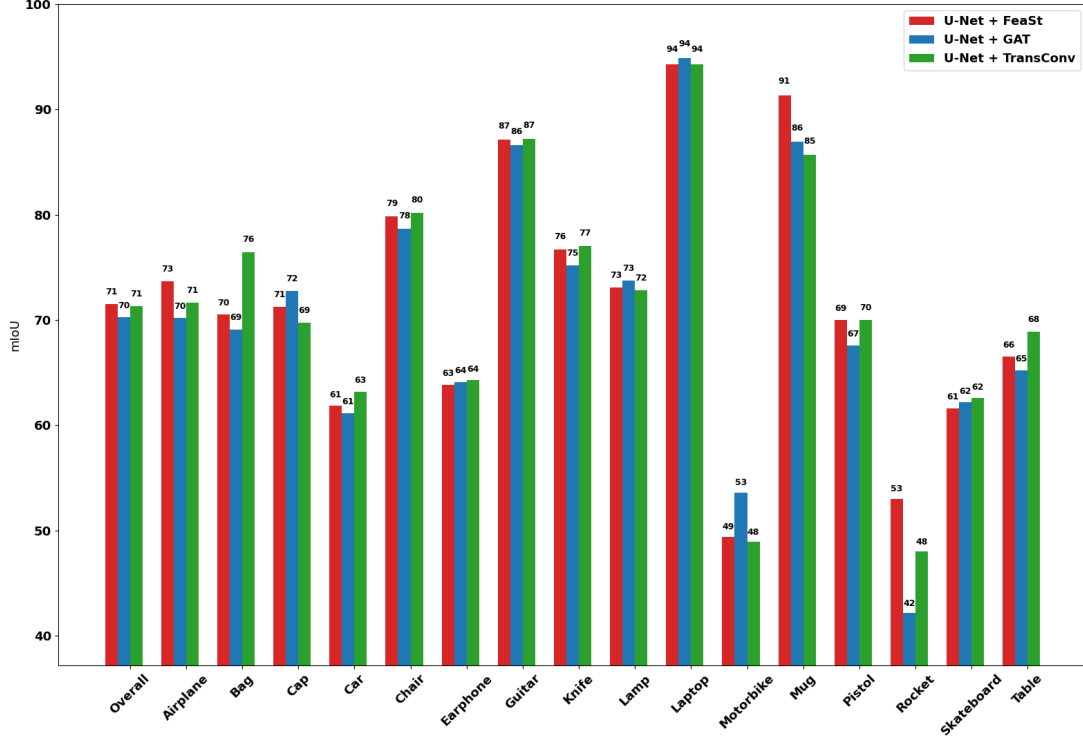


Figure 7.4: Segmentation accuracy in mIoU on the COSEG dataset for U-Net + FeaSt, U-Net + GAT and our work U-Net + TransConv. mIoU values are shown on the y-axis and the categories are on the x-axis. Each bar represents a different model as mentioned in the legend. Green bar represents our work.

has more impact in these models compared to MeshTrans

Table 7.6: Segmentation accuracy in mIoU on the ShapeNet dataset of our approaches with and without Depth Encoding (DE).

	Overall	Airplane	Bag	Cap	Car	Chair	Earphone	Guitar	Knife	Lamp	Laptop	Motorbike	Mug	Pistol	Rocket	Skateboard	Table
MeshTrans	76.19	75.16	71.8	84.06	71.53	84.82	62.14	88.5	77.49	78.11	95.77	58.89	91.21	79.88	53.25	72.68	73.79
MeshTrans + DE	75.78	75.1	70.21	82.92	72.75	82.19	61.81	88.9	77.1	77.91	95.8	57.19	91.16	79.5	55.63	72.17	72.18
FeaStNet + TransConv	77.2	78.1	72.69	80.98	72.19	86.28	62.65	89.01	77.54	79.77	96.04	61.76	92.7	80.41	57.49	71.54	76.13
FeaStNet + TransConv + DE	77.44	78.91	73.13	81.3	71.89	87.68	59.16	88.81	78.24	80.15	95.84	61.19	92.9	81.11	61.22	70.71	76.91
U-Net + TransConv	70.99	71.67	76.45	69.76	63.19	80.19	64.27	87.17	77.05	72.8	94.32	48.91	85.71	70.01	43.01	62.56	68.91
U-Net + TransConv + DE	71.65	71.81	75.18	70.16	64.18	82.04	63.8	87.97	77.35	72.17	96.32	50.18	85.11	70.87	47.12	63.06	69.11

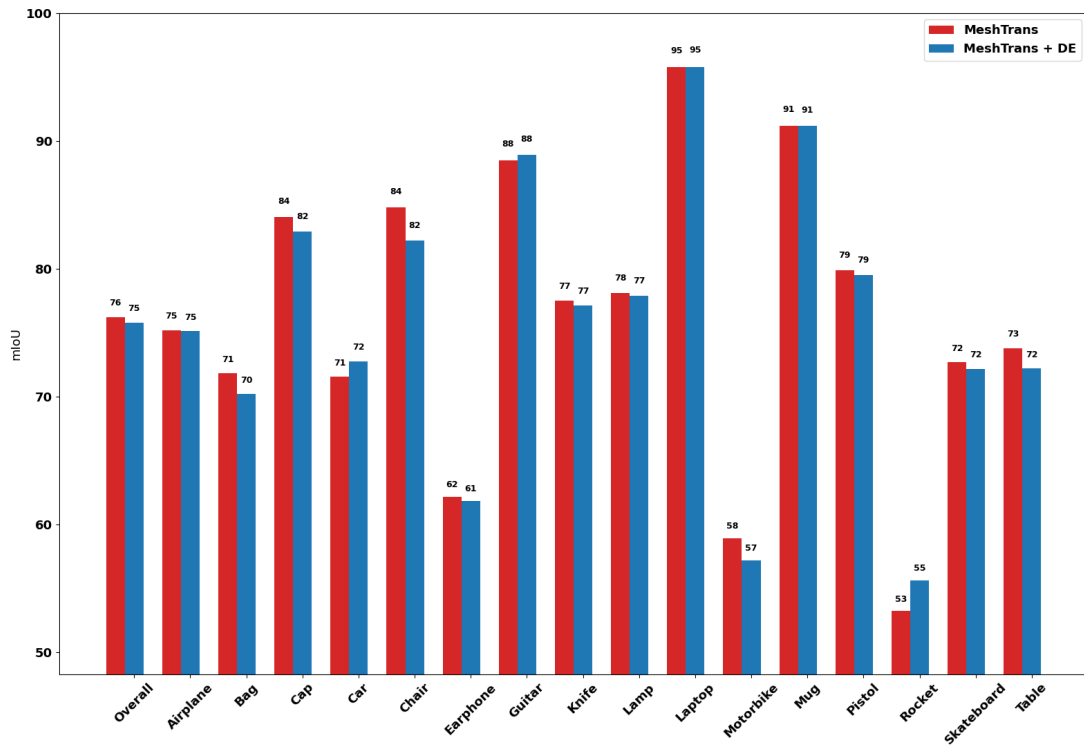


Figure 7.5: Segmentation accuracy in mIoU on the ShapeNet dataset for MeshTrans with and without depth encoding (DE).

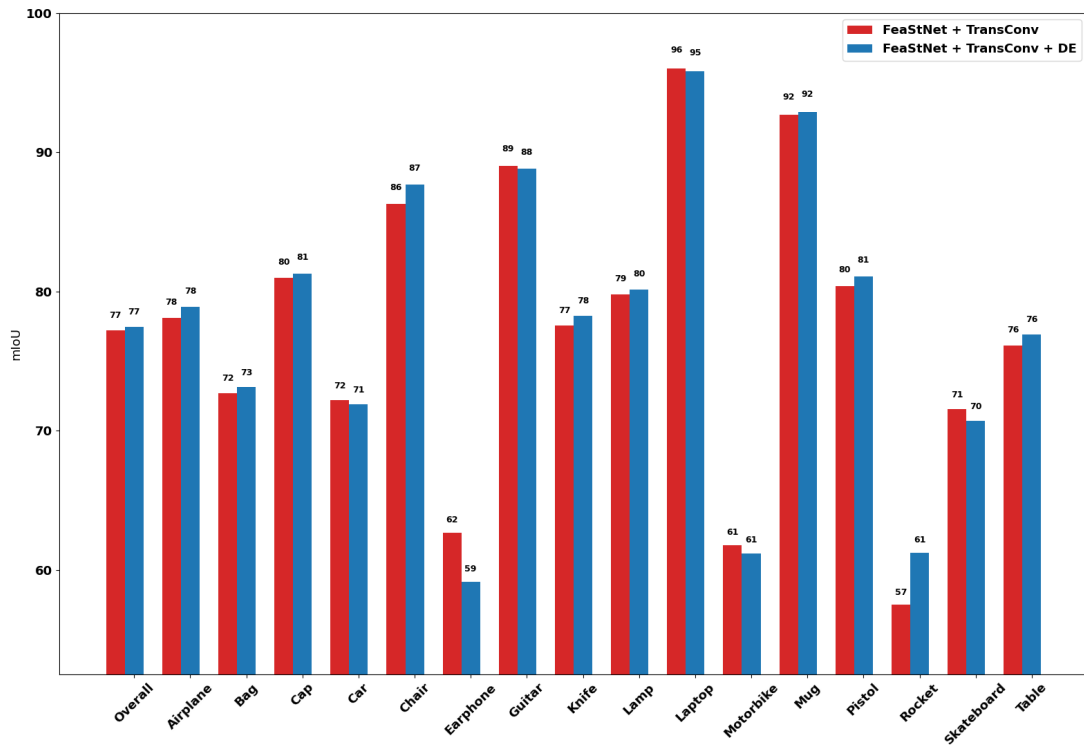


Figure 7.6: Segmentation accuracy in mIoU on the ShapeNet dataset for FeaStNet + TransConv with and without depth encoding (DE).

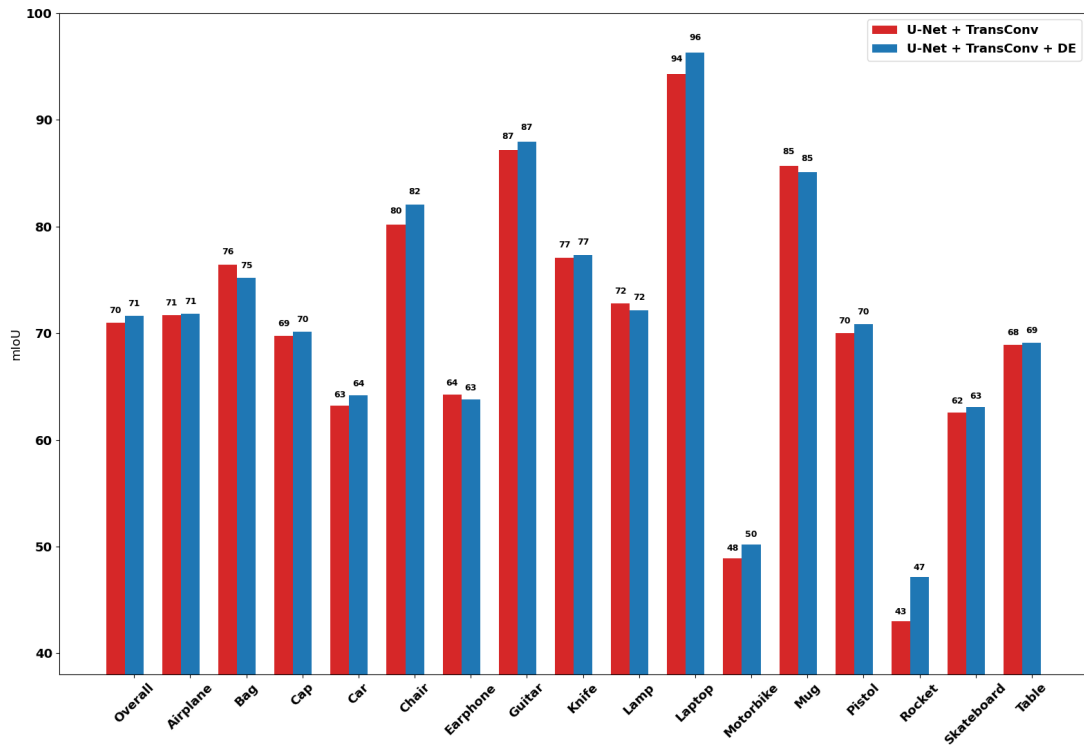


Figure 7.7: Segmentation accuracy in mIoU on the ShapeNet dataset for U-Net + TransConv with and without depth encoding (DE).

8 Future Steps

In this thesis, multiple approaches for the Mesh Segmentation task and extensions to these approaches are discussed. However, there are many other ideas that could be tested to improve the performance of our approaches, but are beyond the scope of this thesis. Such ideas are discussed in this chapter.

8.1 Depth Encoding strategies

Depth Encoding for Mesh Segmentation was discussed in section 5.4 which adds information about the geodesic distance (distance along the graph) to the model input. Similar to positional encoding in transformer network a sinusoid function was used to encode the depth information. However, there can also be other alternatives functions to depth encoding that could better encode depth information.

One idea is to use learning-based depth encodings that can be learned by the model during the training. Right now the depth encoding is added to the input at the very start of the model before the encoder is called, another idea can be to add depth encoding at different stages in the transformer.

Unlike in sentences where the positional value can go up to a 100 in very long sentences, in graphs, the max depth that is used even in atrous convolution is a depth of 3. The depth encoding function can be modified to consider the maximum depth to better distinguish different depths

8.2 Datasets

In this thesis, our approach is tested on two datasets namely COSEG and ShapeNet to show the performance on both point clouds and on 3D meshes. However, both these datasets consisted of 3D models of single objects. In the future, our approach could be tested on more detailed datasets such as ScanNet which consists of 3D scanning of rooms that are

more ricjer in structure and contain more disconnected components compared to single objects.

Also, a combination of meshes and point clouds can be used to train the model to take advantage of both the geodesic distance and the euclidean distance similar to [Sch+20]

8.3 Memory Consumption

One major issue with applying deep learning directly to real-world applications is the GPU memory consumption of a model. Many strategies have been developed to reduce the memory footprint of a model. Although both our approaches (MeshTrans and TransConv) have the model parameter count that is similar to other related works, the memory requirement during an inference is much higher for MeshTrans and TransConv due to creation of the neighbor matrix $H = \mathcal{N}_{h_i} \cup \{h_i\}$. This neighbor matrix is then used to calculate the attention weight matrix $H^T H$ where every neighboring vertex attends to every other vertex. Because every neighbor attends to every other neighbor, in a batched input there is a high chance of repetitive attention weight calculations. To reduce both the number of attention weight calculations and the memory, attention weights between all possible neighbor pairs can be pre-computed and then can be scattered to form the attention weight matrix which takes much less memory compared to the neighbor matrix H .

9 Conclusion

In this thesis, two new approaches to the task of Mesh Segmentation are presented. In the first approach, it is shown how a mesh segmentation task can be thought of as a Machine Translation task and a first of its kind application of transformer [Vas+17] networks to mesh segmentation namely Transformer for Mesh Segmentation (MeshTrans) is introduced.

In our second approach, a novel transformer-based Graph Convolution layer named TransConv is introduced where in a single convolution operation, not only the root vertex but all the neighboring vertices pass information to all other neighboring vertices using the attention mechanism.

Both our approaches are tested on two types of datasets, one a point cloud dataset where the graph data is generated using KNN algorithm and the second a high resolution mesh dataset where the neighbors are decided using the faces of the mesh.

It is shown that our approach to directly use the transformer network for mesh segmentation (Transformer for Mesh Segmentation (MeshTrans)) has a weaker performance in high resolution meshes from the COSEG dataset due to the lack of message propagation and low receptive field, but produces comparable results on point cloud data from the ShapeNet dataset in all the categories. In some categories the MeshTrans performs significantly better than other models, thus showing a lot of potential with further experiments. Our approach to create a graph convolution layer named TransConv outperforms other related works in most categories of both the datasets. In addition to the results from these approaches, the effect of incorporating Depth Encoding into our graph convolution layer is also shown.

List of Figures

1.1	Shown an example of segmentation of a mechanical part. The image is from [Buo+17].	2
1.2	Shown an example of 3D understanding of your environment during autonomous driving. The image is from [Mit19].	2
1.3	Different data structures for 3D data. The images are from [WLX12], [Aga+11] and [Pol10].	3
1.4	An example where meshes can easily convey the distinct identities of the camel joints through geodesic separation, despite their proximity in euclidean space. The image is from [Han+19].	4
3.1	Convolution operations on images-2D (left) vs graph-3D (right). Red node is the root node for which convolution is being calculated. Black edges indicate which nodes are being used in the convolution operation. In images when a 3 X 3 filter is applied all the pixel values in the 3 x 3 patch are used to calculate the final value for the root node. In a graph typically all the neighbors of the root node are used to calculate the value for the root node.	13
3.2	Feature Steered graph convolutional network, where each node in the input patch is associated in a soft manner to each of the M weight matrices based on its features using the weight $q_m(x_i, x_j)$. The image is from [VBV18]. . .	16

3.3	(a) The 1-ring neighbors of the edge e can be ordered as (a,b,c,d) or (c,d, a,b), depending on which face defines the first neighbor. To avoid the ambiguity new set of features are generated by applying simple symmetric functions on each pair (e.g. $\{sum(a, c), sum(b, d)\}$). (b) The input edge feature is a 5- dimensional vector for every edge: the dihedral angle, two inner angles and two edge-length ratios for each face. The edge ratio is between the length of the edge and the perpendicular (dotted) line for each adjacent face. In MeshCNN they sort each of the two face-based features (inner angles and edge-length ratios), thereby resolving the ordering ambiguity and guaranteeing invariance. The images are from [Han+19].	17
3.4	Left: The attention mechanism $a(W.h_i, W.h_j)$ employed by GAT, parametrized by a weight vector, applying a LeakyReLU activation. Right: An illustration of multihead attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain h'_1	19
4.1	Sequence-to-sequence encoder decoder architecture, where the encoder and the decoder is a sequence of RNNs. The encoder consisting of stack of RNNs takes the sequence as an input and generates a final hidden state vector which is then sent to the decoder which again consists of stack of RNNs to predict an output sequence. The image is from [Kos19].	21
4.2	From [BCB16]. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight α_{ji} of the annotation of the j -th source word for the i -th target word in grayscale.	23
4.3	High level architecture of the transformer network.	25
4.4	Architecture of the encoder in the transformer network.	27
4.5	Architecture of the decoder in the transformer network.	28
4.6	Low level architecture of the complete transformer network.	29
4.7	Scaled Dot-Product Attention over $Q \in \mathbb{R}^{N \times d_{qk}}$ and key-value pairs $K \in \mathbb{R}^{M \times d_{qk}}, V \in \mathbb{R}^{M \times d_v}$	31
4.8	Multi-head attention layer from the transformer network over $Q \in \mathbb{R}^{N \times d_{qk}}$ and key-value pairs $K \in \mathbb{R}^{M \times d_{qk}}, V \in \mathbb{R}^{M \times d_v}$	32

4.9	Machine translation task at high level.	34
4.10	Shows inputs to each of the step involved in translating the sentence "How are you?" to spanish. Demonstrates the auto-regressive design of the model.	36
5.1	Shows the graphical representation of the problem statement for this thesis, which is semantic segmentation of 3D meshes. The input on the left hand side is a 3D mesh and the expected output is the semantic segmentation of the 3D mesh.	38
5.2	Architecture of Transformer for Mesh Segmentation, consisting of L encoder layers and L decoder layers. For any vertex i , a neighborhood matrix X_i is created, which is passed through a feed-forward neural network before passing them to the encoder stack. The encoder output along with the root node features are passed through the decoder stack to generate output category probabilities.	40
5.3	Encoder layer operation from the encoder part of the transformer network. .	42
5.4	Two components of Set Transformers that are used in our TransConv operation. .	44
5.5	Final architecture of the Transformer Convolution layer. For any given vertex all the neighboring vertex features (found using the edges) are stacked to form a matrix $H_i \in \mathbb{R}^{(n+1) \times F}$. This matrix is first passed through a feed-forward neural network and then passed through the encoder layer followed by another simple feed-forward neural network. The output is passed on to the Pooling by Multihead Attention layer to generate the final output of the Convolution layer.	46
5.6	Atrous convolution on 2D input using a kernel of width size 3 with rate $r = 2$. The image is from [Prö18].	48
5.7	Atrous convolution supports exponential expansion of the receptive field without loss of resolution or coverage. (a) x_1 is produced from input by a 1-dilated convolution; each element in X_1 has a receptive field of 3×3 . (b) X_2 is produced from X_1 by a 2-dilated convolution; each element in X_2 has a receptive field of 7×7 . (c) X_3 is produced from X_2 by a 4-dilated convolution; each element in X_3 has a receptive field of 15×15 . The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly. The images are from [YK16].	49

5.8	An example of atrous convolution on graph input. For the root vertex (in red) new edges with the vertices at depth one (in yellow) and depth three (in orange) are added to existing edges are added before applying any Graph Convolution.	50
6.1	Overview of a single step in the active co-analysis from [Wan+12]: (a) Start with an initial unsupervised co-segmentation of the input set. (b) During active learning, the system automatically suggests constraints which would refine results and the user interactively adds constraints as appropriate. In this example, the user adds a cannot-link constraint (in red) and a must-link constraint (in blue) between segments. (c) The constraints are propagated to the set and the co-segmentation is refined. The process from (b) to (c) is repeated until the desired result is obtained.	54
6.2	Sample meshes and their labelling from the Shape COSEG (COSEG) dataset. The images are from [Wan+12].	55
6.3	Overview of the ShapeNet part-annotation pipeline used to generate the labelling of 3D models from [Yi+16]. Given the input dataset, an annotation set is selected and UI is used to obtain human labels. The method automatically propagate these labels to the rest of the shapes and then query the users to verify the most confident propagations. Then these verifications are used to improve our propagation technique.	58
6.4	Shows an example of one point cloud for each category from the ShapeNet part-annotation dataset and their labels. The image is from [Yi+16].	59
6.5	Model Architecture of the FeaStNet. $\text{Lin}P$ corresponds to a simple feed-forward neural network with P output channels. $\text{Conv}P$ corresponds to a graph convolution with P output channels. MaxPool is pooling operation over features of each vertex. Global MaxPool is max pooling operation over features of all the vertices. Each of the sublayers are followed by a LeakyRelu (slope=0.1) activation layers.	62
6.6	U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. The image is from [RFB15].	64

6.7	U-net architecture for graphs. $\text{Lin}P$ corresponds to a simple feed-forward neural network with P output channels. $\text{Conv}P$ corresponds to a graph convolution with P output channels. Edge pooling and unpooling are pooling operations explained in section 6.3.3.	65
7.1	Segmentation accuracy in mIoU on the COSEG dataset for FeaStNet + FeaSt, FeaStNet + GAT, FeaStNet + TransConv, MeshTrans. mIoU values are shown on the y-axis and the categories are on the x-axis. Each data point shape represents a different model. The green and the pink bar represents our works TransConv and MeshTrans respectively.	71
7.2	Segmentation accuracy in mIoU on the COSEG dataset for U-Net + FeaSt, U-Net + GAT, U-Net + TransConv. mIoU values are shown on the y-axis and the categories are on the x-axis. Each data point shape represents a different model. Green bar shows results for our graph convolution layer TransConv.	72
7.3	Segmentation accuracy in mIoU on the ShapeNet dataset for FeaStNet + FeaSt, FeaStNet + GAT, FeaStNet + TransConv, MeshTrans. mIoU values are shown on the y-axis and the categories are on the x-axis. Each bar represents a different model as shown in the plot legend. The green and the pink bar represents our works TransConv and MeshTrans respectively. . . .	74
7.4	Segmentation accuracy in mIoU on the COSEG dataset for U-Net + FeaSt, U-Net + GAT and our work U-Net + TransConv. mIoU values are shown on the y-axis and the categories are on the x-axis. Each bar represents a different model as mentioned in the legend. Green bar represents our work.	75
7.5	Segmentation accuracy in mIoU on the ShapeNet dataset for MeshTrans with and without depth encoding (DE).	76
7.6	Segmentation accuracy in mIoU on the ShapeNet dataset for FeaStNet + TransConv with and without depth encoding (DE).	77
7.7	Segmentation accuracy in mIoU on the ShapeNet dataset for U-Net + TransConv with and without depth encoding (DE).	78

List of Tables

5.1	Shows the list of hyper parameters and their descriptions in a MeshTrans network.	39
5.2	Shows for each of the hyper parameters in a single Transformer Convolution layer, the names and descriptions.	47
6.1	Shows few of the publicly available datasets for the task of 3D segmentation. The first column is the name of the dataset, followed by the data format provided by the dataset, followed by the high level category of the 3D data provided by the dataset, followed by references to each of the datasets. . . .	53
6.2	COSEG: Starting from the first column shows the name of the categories, number of 3D models for that category, number of 3D models used for training and for testing.	56
6.3	COSEG: Shows for each category in Shape COSEG dataset, the number of labels.	57
6.4	ShapeNet: Starting from the first column shows the name of the categories, number of 3D models for that category, number of 3D models used for training, used for validation and used for testing.	60
6.5	Shows the number of labels for each category and their names for better understanding. Unnamed label refers to a label that exists in the annotations but does not have a name that can describe it.	61
7.1	Shows different configurations of models and graph convolution layers that are tested in our thesis. The rows are the three different models discussed in section 6.3 and the columns are different graph convolution layers used inside the model architecture. For MeshTrans, there are no graph convolution layers that can be configured.	70

7.2	Segmentation accuracy in mIoU on the COSEG dataset for FeaStNet + FeaSt, FeaStNet + GAT, FeaStNet + TransConv, MeshTrans.	71
7.3	Segmentation accuracy in mIoU on the COSEG dataset for U-Net + FeaSt, U-Net + GAT and U-Net + TransConv (ours) model configurations.	72
7.4	Segmentation accuracy in mIoU on the ShapeNet dataset for FeaStNet + FeaSt, FeaStNet + GAT, FeaStNet + TransConv (Ours) and MeshTrans (Ours) model configurations.	73
7.5	Segmentation accuracy in mIoU on the ShapeNet dataset for U-Net + FeaSt, U-Net + GAT and U-Net + TransConv (ours) model configurations.	74
7.6	Segmentation accuracy in mIoU on the ShapeNet dataset of our approaches with and without Depth Encoding (DE).	75

Bibliography

- [Aga+11] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski. “Building Rome in a Day.” In: *Commun. ACM* 54.10 (Oct. 2011), pp. 105–112. ISSN: 0001-0782. DOI: 10.1145/2001269.2001293.
- [Arm+16] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. “3D Semantic Parsing of Large-Scale Indoor Spaces.” In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 1534–1543. DOI: 10.1109/CVPR.2016.170.
- [AT16] J. Atwood and D. Towsley. *Diffusion-Convolutional Neural Networks*. 2016. arXiv: 1511.02136 [cs.LG].
- [BCB16] D. Bahdanau, K. Cho, and Y. Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].
- [BKH16] J. L. Ba, J. R. Kiros, and G. E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].
- [Buo+17] F. Buonomici, M. Carfagni, R. Furferi, L. Governi, A. Lapini, and Y. Volpe. “Reverse Engineering of Mechanical Parts: a Template-Based Approach.” In: *Journal of Computational Design and Engineering* 5 (Nov. 2017). DOI: 10.1016/j.jcde.2017.11.009.
- [CGF09] X. Chen, A. Golovinskiy, and T. Funkhouser. “A Benchmark for 3D Mesh Segmentation.” In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 28.3 (Aug. 2009).
- [Cha+15] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

- [Che+17] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. 2017. arXiv: 1606.00915 [cs.CV].
- [Cho+14] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].
- [Chu+14] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE].
- [CMZ19] S. Chen, K. Ma, and Y. Zheng. “Med3D: Transfer Learning for 3D Medical Image Analysis.” In: *CoRR* abs/1904.00625 (2019). arXiv: 1904.00625.
- [Dai+17] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. “ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes.” In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*. 2017.
- [Dai+18] H. Dai, Z. Kozareva, B. Dai, A. Smola, and L. Song. “Learning Steady-States of Iterative Algorithms over Graphs.” In: ed. by J. Dy and A. Krause. Vol. 80. *Proceedings of Machine Learning Research*. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 1106–1114.
- [Die19] F. Diehl. *Edge Contraction Pooling for Graph Neural Networks*. 2019. arXiv: 1905.10990 [cs.LG].
- [DT20] M. Denninger and R. Triebel. “3D Scene Reconstruction from a Single Viewport.” In: Aug. 2020.
- [Fal19] W. Falcon. “PyTorch Lightning.” In: *GitHub*. Note: <https://github.com/PyTorchLightning/pytorch-lightning> 3 (2019).
- [FL19] M. Fey and J. E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric.” In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.
- [Gil+17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG].

- [GM10] C. Gallicchio and A. Micheli. “Graph Echo State Networks.” In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. 2010, pp. 1–8. DOI: 10.1109/IJCNN.2010.5596796.
- [Han+19] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. “MeshCNN.” In: *ACM Transactions on Graphics* 38.4 (July 2019), pp. 1–12. ISSN: 1557-7368. DOI: 10.1145/3306346.3322959.
- [He+15] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].
- [He+18] K. He, G. Gkioxari, P. Dollár, and R. Girshick. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV].
- [HS97] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory.” In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.
- [HYL18] W. L. Hamilton, R. Ying, and J. Leskovec. *Inductive Representation Learning on Large Graphs*. 2018. arXiv: 1706.02216 [cs.SI].
- [Iso+18] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004 [cs.CV].
- [Kal+17] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri. “3D Shape Segmentation with Projective Convolutional Networks.” In: *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*. 2017.
- [KB17] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].
- [KHS10] E. Kalogerakis, A. Hertzmann, and K. Singh. “Learning 3D Mesh Segmentation and Labeling.” In: *ACM Transactions on Graphics* 29.3 (2010).
- [Kos19] S. Kostadinov. *Understanding Encoder-Decoder Sequence to Sequence Model*. Nov. 2019.
- [KW16] T. N. Kipf and M. Welling. *Variational Graph Auto-Encoders*. 2016. arXiv: 1611.07308 [stat.ML].
- [KW17] T. N. Kipf and M. Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

- [Law+17] F. J. Lawin, M. Danelljan, P. Tosteberg, G. Bhat, F. S. Khan, and M. Felsberg. “Deep Projective 3D Semantic Segmentation.” In: *Computer Analysis of Images and Patterns*. Ed. by M. Felsberg, A. Heyden, and N. Krüger. Cham: Springer International Publishing, 2017, pp. 95–107. ISBN: 978-3-319-64689-3.
- [Lee+19] J. Lee, Y. Lee, J. Kim, A. R. Kosiosek, S. Choi, and Y. W. Teh. *Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks*. 2019. arXiv: 1810.00825 [cs.LG].
- [Li+17] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. *Gated Graph Sequence Neural Networks*. 2017. arXiv: 1511.05493 [cs.LG].
- [Li+18] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. *PointCNN: Convolution On \mathcal{X} -Transformed Points*. 2018. arXiv: 1801.07791 [cs.CV].
- [LPM15] M.-T. Luong, H. Pham, and C. D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. arXiv: 1508.04025 [cs.CL].
- [MGG12] D. Merrill, M. Garland, and A. Grimshaw. “Scalable GPU Graph Traversal.” In: *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPOPP ’12. New Orleans, Louisiana, USA: Association for Computing Machinery, 2012, pp. 117–128. ISBN: 9781450311601. DOI: 10.1145/2145816.2145832.
- [Mic09] A. Micheli. “Neural Network for Graphs: A Contextual Constructive Approach.” In: *IEEE Transactions on Neural Networks* 20.3 (2009), pp. 498–511. DOI: 10.1109/TNN.2008.2010350.
- [Mit19] D. Mitrev. *Audi Released Autonomous Driving Dataset A2D2*. Sept. 2019.
- [Mo+19] K. Mo, S. Zhu, A. Chang, L. Yi, S. Tripathi, L. Guibas, and H. Su. “PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding.” In: (2019).
- [NAK16] M. Niepert, M. Ahmed, and K. Kutzkov. *Learning Convolutional Neural Networks for Graphs*. 2016. arXiv: 1605.05273 [cs.LG].
- [Pan+19] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang. *Adversarially Regularized Graph Autoencoder for Graph Embedding*. 2019. arXiv: 1802.04407 [cs.LG].

- [Pas+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035.
- [Pol10] Polygon mesh. *Polygon mesh — Wikipedia, The Free Encyclopedia*. [Online; accessed 12-December-2020]. 2010.
- [Prö18] P.-L. Pröve. *An Introduction to different Types of Convolutions in Deep Learning*. Feb. 2018.
- [Qi+17a] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. arXiv: 1612.00593 [cs.CV].
- [Qi+17b] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. 2017. arXiv: 1706.02413 [cs.CV].
- [Ren+16] S. Ren, K. He, R. Girshick, and J. Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].
- [RFB15] O. Ronneberger, P. Fischer, and T. Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].
- [SC08] A. Shamir and D. Cohen-Or. “Consistent mesh partitioning and skeletonisation using the shape diameter function.” In: *The Visual Computer* 24 (Apr. 2008), pp. 249–259. DOI: 10.1007/s00371-007-0197-5.
- [Sca+09] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. “The Graph Neural Network Model.” In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.
- [Sch+20] J. Schult, F. Engelmann, T. Kontogianni, and B. Leibe. *DualConvMesh-Net: Joint Geodesic and Euclidean Convolutions on 3D Meshes*. 2020. arXiv: 2004.01002 [cs.CV].
- [SVL14] I. Sutskever, O. Vinyals, and Q. V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: 1409.3215 [cs.CL].

- [TPL20] M. Tan, R. Pang, and Q. V. Le. *EfficientDet: Scalable and Efficient Object Detection*. 2020. arXiv: 1911.09070 [cs.CV].
- [Tu+18] K. Tu, P. Cui, X. Wang, P. S. Yu, and W. Zhu. “Deep Recursive Network Embedding with Regular Equivalence.” In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 2357–2366. ISBN: 9781450355520. DOI: 10.1145/3219819.3220068.
- [Vas+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].
- [VBV18] N. Verma, E. Boyer, and J. Verbeek. *FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis*. 2018. arXiv: 1706.05206 [cs.CV].
- [Vel+18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].
- [Wan+12] Y. Wang, S. Asafi, O. van Kaick, H. Zhang, D. Cohen-Or, and B. Chen. “Active Co-Analysis of a Set of Shapes.” In: *ACM Trans. Graph.* 31.6 (Nov. 2012). ISSN: 0730-0301. DOI: 10.1145/2366145.2366184.
- [Wan+13] Y. Wang, M. Gong, T. Wang, D. Cohen-Or, H. Zhang, and B. Chen. “Projective Analysis for 3D Shape Segmentation.” In: *ACM Trans. Graph.* 32.6 (Nov. 2013). ISSN: 0730-0301. DOI: 10.1145/2508363.2508393.
- [Wan+19] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. *Dynamic Graph CNN for Learning on Point Clouds*. 2019. arXiv: 1801.07829 [cs.CV].
- [WCZ16] D. Wang, P. Cui, and W. Zhu. “Structural Deep Network Embedding.” In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1225–1234. ISBN: 9781450342322. DOI: 10.1145/2939672.2939753.
- [WLX12] Z. Wang, H. Liu, and T. Xu. “Real-Time Plane Segmentation and Obstacle Detection of 3D Point Clouds for Indoor Scenes.” In: vol. 7584. Oct. 2012. ISBN: 978-3-642-33867-0. DOI: 10.1007/978-3-642-33868-7_3.

- [Wu+20] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. “A Comprehensive Survey on Graph Neural Networks.” In: *IEEE Transactions on Neural Networks and Learning Systems* (2020), pp. 1–21. ISSN: 2162-2388. DOI: 10.1109/tnnls.2020.2978386.
- [Yi+16] L. Yi, V. G. Kim, D. Ceylan, I.-C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, and L. Guibas. “A Scalable Active Framework for Region Annotation in 3D Shape Collections.” In: *ACM Trans. Graph.* 35.6 (Nov. 2016). ISSN: 0730-0301. DOI: 10.1145/2980179.2980238.
- [YK16] F. Yu and V. Koltun. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2016. arXiv: 1511.07122 [cs.CV].
- [ZWC19] W. Zhu, X. Wang, and P. Cui. “Deep Learning for Learning Graph Representations.” In: *Studies in Computational Intelligence* (Oct. 2019), pp. 169–210. ISSN: 1860-9503. DOI: 10.1007/978-3-030-31756-0_6.