Computer Science
Carnegie Mellon University in Qatar

SCS Honors Undergraduate Research Thesis

Spring 2024

SampleLapNet: A Learnable Laplacian Approach for Task-Agnostic Point Cloud Downsampling

by

Diram Tabaa

Advisor

Professor Gianni Di Caro

Carnegie Mellon University Qatar

Contents

Chapter 1

Introduction

In recent years, we have witnessed greater integration of autonomous and semi-autonomous agents in our world. From self-driving cars to remote sensing drones [24], autonomous agents have developed greater environment awareness due to advancements both at the software level (e.g. AI-based algorithms), as well as at the hardware level (e.g. higher-fidelity sensors). In particular, many autonomous agents are now equipped with sensors that are capable of capturing three-dimensional (3D) information, whether in the form of depth maps with RGB-D cameras, or point cloud data with Light Detection and Ranging (LiDAR) sensors. Due to the widespread integration of LiDAR sensors in modern devices, point clouds became the standard when it comes to collecting 3D data from the environment in real-time.

Yet beyond the hardware, advancements in the fields of computer vision and deep learning have made it practical to utilize 3D data in environment sensing & awareness scenarios. As an example, consider the task of semantic segmentation, which entails assigning distinct labels to semantically equivalent parts of the data - hence segmenting it. Such semantic definitions range from object-level semantics (doors, tables, cars, etc..) to part-level semantics (door handle, table-top, car wheels), and even instance-level semantics (door$_1$, table$_7$, car$_8$ [6]. Like their two-dimensional counterpart (images & vectors), it was quite difficult to achieve acceptable results on such tasks before the dawn of Deep Neural Networks. Unlike images, however, the field of deep learning on 3D data generally, and on point clouds specifically, has considerably lagged behind in the past for reasons that we will detail in section 1.3. Nevertheless, current research trends in the field show promising results and provide a significant room for exploration, as we shall see in this project.

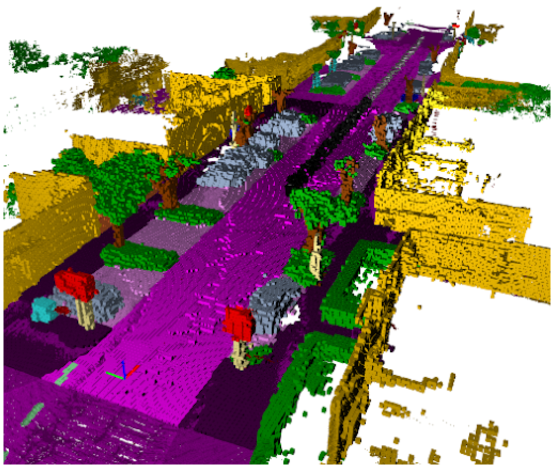


Figure 1.1: Example of 3D data collected from ranging sensors on board a car. Different colors in the scene indicate different semantic labels

1.1 Point Cloud Data and its Properties

Point Clouds are a popular representation of 3D information due to their versatility and proximity to the raw data, making them the ideal choice for agents with low computational resources, the likes of drones and autonomous vehicles. Point Clouds are computationally an unordered set of 3D vectors, where each vector represents a point in space. Points in a Point Cloud could also contain other features, such as color or vertex normals, that provide more information about the captured scene. In order for Point Clouds to accurately represent an object or a scene, they need to be significantly denser relative to other 3D representations, due to the lack of connectivity information.

Another important feature of Point Clouds is the lack of a uniformity constraint. Unlike images, where each pixel has a fixed "position", points in a Point Cloud could be distributed arbitrarily, which could result in certain regions being dense, and other regions being sparse. This feature introduces a challenge of inferring continuity, since it is impossible to determine where an object in a scene "begins" and "ends". The table below summarizes the features of point clouds in comparison to Polygonal Meshes and Voxels:

Spatial Feature	Point Clouds	Polygonal Meshes	Voxels
Density	Variable	Variable	Uniform
Connectivity	None	Faces & Edges	None
Neighbourhood	Not Defined	Edge-Defined	Index-Defined

Table 1: Summary of spatial features in different 3D data representations

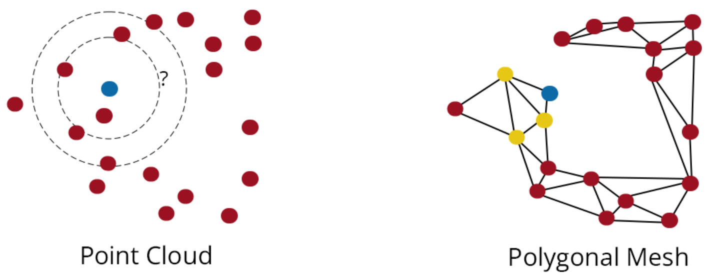


Figure 1.2: Unlike polygonal meshes, point clouds do not have a defined sense of neighbourhood

1.2 Point Cloud Laplacian

Despite the limitations of such spatial features, many algorithms have been developed over the years to perform geometric operations on Point Clouds akin to other 3D data representations like meshes. A great depth of this work is covered under the field of Discrete Differential Geometry (DDG), which tackles the problem of converting higher-order differentiable operations into discrete computations that can be linearly estimated with Linear Algebra libraries [8].

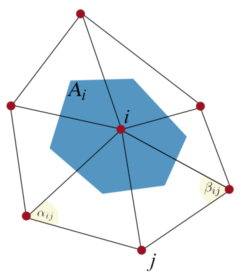
Amongst such differentiable operations is an important operator known as the Laplace-Beltrami Operator (Δ), or more simply the laplacian in Euclidean space. This operator, at a given point in the domain, measures the signed deviation of a function from its local neighbourhood. In Euclidean space, this translates to the divergence of the gradient vector at that point.

Definition 1. *The Laplace–Beltrami operator*

$$\textit{Let } f : M \to \mathbb{R}^n \quad \textit{where } M \textit{ is a manifold (locally-euclidean) space}$$
$$\Delta f = Div(\nabla f) \left(= \sum_{i=1}^{m} \frac{\partial f}{\partial x_i^2} \quad \textit{if } M = \mathbb{R}^m \right)$$

In the discrete mesh setting, the laplacian can be represented as the product of two $n \times n$ matrices, M^{-1} and L. These matrices are intrinsic to a given mesh, and are invariant of the function applied. Matrix M is a diagonal matrix known as the mass matrix, where each diagonal value represents the "area"/"volume" of a vertex. Matrix L is a positive-semidefinite matrix known as the laplacian matrix, that encodes the influence of other points on the value of the laplacian, and is such a symmetric matrix.

If $f : V \to \mathbb{R}$ is a function that maps each point to a scalar function, and if x is a vector of points, then $u = f(x)$ is a vector of function values. Then the laplacian of f is simply $M^{-1}Lu$. An important result can be obtained by considering the laplacian of the identity mapping, namely a vector function that maps each point to its coordinate values. The result of this would be an $n \times 3$ vector, representing the laplacian value in each coordinate direction, this provides us with a measure of "divergence" of each point from its neighbourhood with respect of each of the coordinates.



$$(\Delta u)_i = \frac{1}{2A_i} \sum_{j \in N(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(u_i - u_j)$$

$$L_{ij} = \begin{cases} \frac{1}{2} \sum\limits_{i \sim j} (\cot \alpha_j + \cot \beta_j), & \text{if } i = j. \\ -\frac{1}{2} (\cot \alpha_j + \cot \beta_j), & \text{if } i \sim j. \\ 0, & \text{otherwise} \end{cases}$$

Figure 1.3: Computing the laplacian on a manifold triangle mesh. Similar discretizations exist for other representations

In practice, this operator is useful in many physical and geometric models, such as heat diffusion, wave propagation, and curvature flow [8]. While discretizing the laplacian is straightforward with meshes due to the explicit connectivity data present at each vertex, the same is not true for the case of point clouds. Workarounds for this lack of connectivity in Point Clouds often attempt to exploit nearest-neighbour algorithms to estimate a local neighbourhood for each point [13]. However, this makes the operator much more computationally expensive, since extracting local neighbourhood of a point is not constant time in Point Clouds.

1.3 Learning on Point Cloud Data and its Challenges

As mentioned earlier, the use of point cloud data has become widespread in many learning-based applications due to the ease of collection and processing. However, point cloud data poses a number of challenges for learning-based tasks due to the properties we discussed in the previous section. In this section we discuss the most important challenges faced when attempting to train Deep Learning models on point cloud data.

- **Neighbourhood Aggregation:** unlike images, graphs, or meshes, point clouds lack any defined notion of connectivity or neighbourhood. Instead, neighbourhood is vaguely inferred based on distance measurements subject to a manually selected limits, either in terms of maximum radius or number of neighbours. This makes convolutions and similar forward-aggregation layers both non-generalizable as well as computationally expensive, due to the overhead of computing a point neighbourhood. While this issue can be minimized in training by precomputing all closest neighbours for all training instances, the same doesn't carry over during inference. This introduces a bigger trade-off between model size and inference time, and thus between accuracy and efficiency in real-time applications.

- **Permutation Invariance:** Point Clouds are geometrically invariant with respect to permutations of vertex indices. Neural networks, however, operate on vectors and tensors, which introduces non-invariance with respect to feature ordering. In other words, geometrically identical point clouds that are permutations of each other would be interpreted as different inputs by a neural network, as shown in Figure 1.3. This problem is often resolved by introducing an embedding component before the first layer, but such an embedding increases the depth of the network, and hence its time and memory bounds.

- **Memory Constraints:** As discussed earlier, point clouds have very low information density per unit data. This makes training on point clouds memory consuming, due to both the memory needed to store point cloud tensors, as well as the model weights. This implies that training on point cloud data is not only computationally-costly, but is also greatly limited by available memory on a GPU.

- **Limited Labelled Data:** While various point cloud/mesh datasets exist, with various task benchmarks, there is still a limitation on the number of available labelled training data for point clouds. Self-supervised learning has been used to address this issue, but its use is still in its early stages [23] and is beyond the scope of this research.

It is important to note that all the aforementioned challenges are aggravated in the case of point cloud sequences.In particular, exploiting temporal correlation in point cloud sequences is not as straightforward as in videos due to permutation invariance, which removes inter-frame point matching guarantees.This incurs models additional computation costs due to neighbourhood search operations needed to match "close" points across frames.

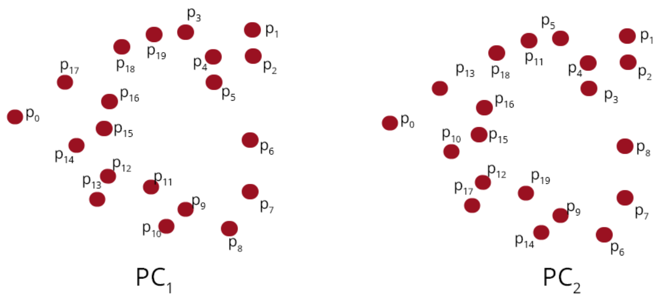


Figure 1.4: While PC_1 and PC_2 are geometrically identical, to a neural network they represent two different inputs due to point cloud serialization into tensors

1.4 Point Cloud Downsampling and its Limitations

Many of the challenges discussed in the earlier section can be resolved by addressing the limitations of point cloud data itself instead of deep learning models. In particular, reducing the number of points in a point cloud, a process known as downsampling, provides a larger margin for deeper and/or wider neural networks. This allows neural networks to generalize better and/or extract more point features respectively while staying within the limits of memory/time-bounds.

Nevertheless, downsampling point clouds in itself is not a straightforward task. Naive methods like uniform or random downsampling do not account for the non-uniform distribution of point clouds, which can lead to an asymmetric loss of information. Even more informed methods that downsample based on point distribution do not take into account information distribution. Figure 1.5 illustrates the issue.

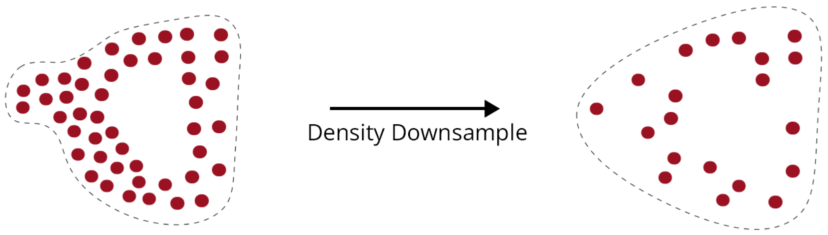


Figure 1.5: Geometry-agnostic downsampling results in loss of spatial informations in point clouds

Moving towards information-aware point cloud downsampling, various methods exist that downsample point clouds while minimizing information loss. A class of such methods exploit geometric measures, such as the Laplacian, to assign an 'importance' score to each point, and select points based on such scores. While such methods are task-agnostic, they pose computational limitations due to the cost of accurately computing those measures on point clouds. Alternatively, learning-based methods have been incorporated that update their selection criteria based on the accuracy of the trained downstream task [9], while such methods lead to quicker inference times due to the streamlined pipeline of layers, they remain limited to a specific task they are trained on, and hence are not generalizable.

1.5 Problem Statement & Contribution

Since deep learning models on point clouds have been greatly optimized over the past few years, especially within the size limits of benchmark datasets, a more realistic goal is to improve inference-time on point clouds by downsampling the point clouds before passing them to the network while maintaining comparable accuracy, or vice versa. However, as we have seen earlier, it is not clear whether or not information-aware downsampling could be achieved without relying on a downstream task to drive the gradients in the direction that optimally select the relevant points for the given task.

In this research, we investigate this problem by proposing a task-agnostic, learnable downsampling network that is capable of selecting points based on geometric intuitions of importance. We incorporate the laplacian as indirect metric of importance by learning to predict which points are more likely to have a higher laplacian norm. Formally, the goal of the project is:

Design and train a neural network capable of predicting the relative discrete distribution of ***identity-function laplacian*** *norms in a point cloud, then utilize this network to generate predictions that guide downsampling*

This project is divided into 3 parts. (1) designing and implementing a neural network capable of estimating the discrete distribution of laplacian norms on a training dataset. (2) Incorporating the pre-trained laplacian network in a standard model for the benchmark task of semantic segmentation (3) Evaluating the applicability of the laplacian network for point cloud sampling as a trade-off between inference time and accuracy.

As far as we know, this project is the first of its kind to use learnt-laplacian values to downsample point clouds. A single research paper [10] has been produced before that makes use of the learnt laplacian idea, but its application has been in a different context, and the methodology involves a nearest neighbour search, which disqualifies it from our application.

Chapter 2

Related Work

2.1 Deep Learning on Point Clouds

Over the years, numerous models have been proposed to perform learning on point cloud data. One of the foundational models, PointNet [4] introduced in 2017, put forth the concept of applying pointwise Multi-layer Perceptrons (MLP) with shared weights. Subsequent models like PointNet++ [12] and SRINet [15] expanded on this fundamental idea, incorporating shared MLPs and structured pooling to aggregate results. While these models were adept in specific contexts such as object classification, they fell short in grasping geometric relationships due to the intrinsically disconnected and geometrically non-correlated pooling methods they employ, such as max and mean pooling.

Inspired by the success of convolutional neural network (CNN) architectures - the likes of VGG [14] and ResNet [7] - in the context of learning on Images, many architectures have attempted to incorporate the concept of convolution on 3D data. Such architectures include PointCNN [11], ConvPoint [3], and KPConv [16]. However, unlike images, convolution is not straightforward to define on 3D data, and potential computationally expensive depending on the representation. Additionally, defining convolution as a strict kernel on a subset of the input potentially limits the learning capabilities of the network due to the irregularity in the data density of certain 3D representations.

More recently, the infamous transformer architecture, which has revolutionized Natural Language Processing, has made its way to the research space of point cloud learning. In particular, architectures like PCT and Point Transformer [25] have demonstrated the applicability of the self-attention mechanism to point cloud data, achieving state-of-the-art performance relative to Linear and Convolutional neural network architectures on tasks like classification and semantic segmantation. Since then, many papers have been published that incorporate the idea of a point cloud transformer, most relevant to us amongst these architectures is Octformer [19], which utilizes an octree data structure to perform attention, and thus by reducing the asymptotic performance of the model for large point clouds. Because our goal is to work with large point clouds, we opt for Octformer as the starting point for our research.

2.2 Learning-based Sampling methods

There has been significant work done to implement differentiable, learning-based down-sampling layers to reduce the number of points in a point cloud data. The motivation behind a learning-based downsampling layer is the ability to fine-tune downsampling for specific downstream tasks e.g. classification or semantic segmentation. One of the earlier works to introduce a differentiable downsampling layer is SampleNet [9]. They achieve this differentiable downsampling by passing the features to a linear layer that reduces the dimensionality of the input, and then for each feature vector of the output, perform a 'soft-projection' operation to the nearest-neighbours of the output feature in the original point cloud. this soft-projection is controlled by a learnable temperature parameter that governs the locality of the neighbourhood. This differentiability means that the whole network can be trained end-to-end, and hence leads to sampling results that are fine-tuned to the needs of the downstream task.

A more recent work in the field of learnable downsampling techniques is APES [21], which is an attention-based edge sampling network. The authors make use of self-attention on point cloud as a measure of "variability" within a neighbourhood (locally or globally) and selecting the points that provide the greatest such variability, which would likely suggest that these points are edge points. Once the self-attention matrix is computed, the columns are summed up, and the points with the largest M column sums are selected. As with the case of SampleNet, this downsampling forward computation is differentiable, and hence allows the overall network to be end-to-end differentiable. With their downsampling technique, they were able to achieve state-of-the-art performance on ModelNet40 classification benchmark, which suggests that not only downsampling is crucial to reduce the computation overhead, but also positively influences the overall accuracy of the model, as it allows the downstream task to focus on the more important spatial features.

Chapter 3

SampleLapNet Architecture

3.1 Overview

In this chapter, we explore the architecture of SampleLapNet, organizing our discussion into the following sections: (1) Preprocessing of point cloud data for Laplace label targets. (2) The architecture of the base Laplacian label predictor. (3) Integration of SampleLapNet with frozen weights into a semantic segmentation model for downsampling. (4) A discussion of the factors influencing our architectural choices.

3.2 Point Cloud preprocessing

In Chapter 1, we touched upon the mathematical basis of the laplacian operator, and the intuitive meaning in terms of average "divergence" away from a local neighbourhood. In this section, we discuss how we incorporate that concept into our architecture.

Given our objective of labeling points in a point cloud based on geometric features, the discrete Laplacian emerges as a fitting metric to employ. Yet, directly learning the discrete Laplacian proves challenging due to the quadratic nature of the Laplace matrix. However, this doesn't imply impossibility, particularly if one makes use of the sparse characteristics of the Laplace matrix to mitigate this non-linearity. For our specific case, directly estimating the Laplace matrix isn't deemed worthwhile since our aim is solely to learn a discrete measure of significance.

A more effective approach to our problem entails predicting the interval within which each point's Laplacian norm lies. Essentially, this involves assigning a label to each point based on the most probable interval for its Laplacian norm. This transforms the problem into a segmentation task, where original labels are pre-computed prior to training. This pre-computation involves manually computing the Laplacians (a process notably slow for large point clouds) and then partitioning the distribution into intervals. Further discussion on this process is provided in Chapter 4. For now, it's important to understand that the task resembles segmentation, albeit with geometry-based labels rather than semantics-based ones.

3.3 SampleLapNet: the laplacian label predictor

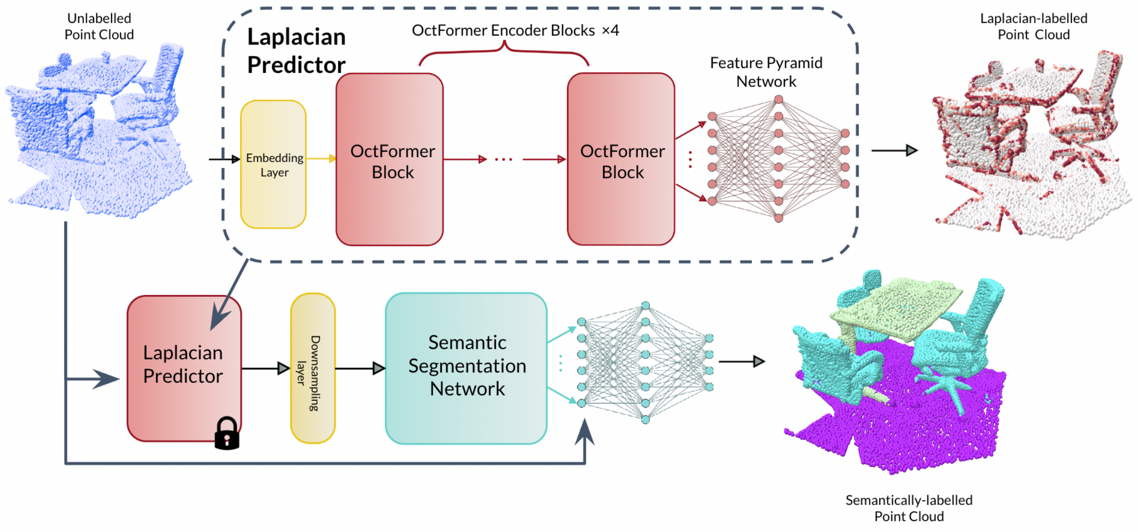


Figure 3.1: An overview of the SampleLapNet Architecture

Our SampleLapNet builds upon the transformer architecture [18]. In particular, we have adopted OctFormer [19], an octree-based transformer architecture for learning on point clouds, with minor modification to train for laplacian labels instead of semantic labels. Before introducing the OctFormer architecture, we provide a brief overview of transformers.

3.3.1 Transformers

Transformers are encoder-decoder models which have seen success through their encoder self-attention mechanism. Transformers encode the input sequence into a latent vector representation that is fed into a decoder to generate/extract features required for a task. In our case, we train the decoder to generate the laplacian vector.

Transformers make use of attention between the encoder and decoder outputs to allow decoder to attend over positions in the input sequence. Relating back to our task, this allows the decoder to attend to the correct "neighbourhood" in the process of generating the laplacian values.

Attention, as the name emplies, is a weighting of values V, based on a query Q made on a keys K. In practice, this attention is computed through a mechanism known as scaled dot-product attention, shown in equation 3.1

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.1}$$

The attention mechanism itself does not involve learnt parameters. On the other hand, queries, keys and values are a result of a linear projection of input features. This becomes clearer in the case of multi-head attention, where input vectors are linearly projected to produce multiple queries, keys and vectors. Each linear projection is its own learnt matrix, allowing different 'heads' to attend to different features. Equation 3.2 illustrates this mechanism:

$$\begin{aligned} &MultiHeadAttention(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^O \\ &\text{where } head_i = Attention(QW_i^k, KW_i^k, VW_i^v) \end{aligned} \tag{3.2}$$

In self-attention, all of the three inputs to the attention module is the input sequence itself. Self-attention allows the encoder to relate different positions in an input sequence as it moves towards a latent representation. A typical encoder block combines this self-attention with a feed-forward network. Multiple encoder blocks are combined together to form an encoder layer.

3.3.2 OctFormer and SampleLapNet

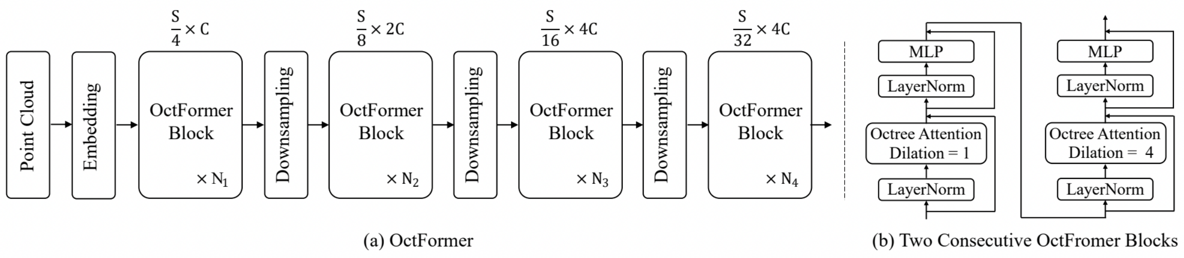


Figure 3.2: OctFormer Architecture [19]. Note that the feature downsampling layers are convolution operations that are unrelated to our point cloud downsampling.

OctFormer is an octree-based transformer architecture. Like its classical counterpart, OctFormer builds upon the concept of (self-)attention to learn a weighting of the sum of values. Unlike a classical transformer, however, the octformer architecture implements an improved octree-based attention that reduces the asymptotic complexity of attention. If we let N be the number of points in a point cloud. Clearly, in a classical transformer, the asymptotic complexity of attention is $O(N^2)$, which for a point cloud of size $> 100k$ is simply too slow.

Octformer alleviates this by constructing an octree structure on the point cloud, where the leaf nodes are either empty, or store a non-negative number of features that correspond to the points within the coordinate limits (voxel) of the leaf node. After building the octree, the nodes at the same depth are sorted according to their shuffled keys, the the shuffle key for the octree node with integer coordinates (x, y, z) is computed as below:

$$Key(x, y, z) = x_1y_1z_1x...x_dy_dz_d \quad \text{where } d \text{ is the depth of the octree} \tag{3.2}$$

Once this sorting is achieved, and all the non-empty nodes are filtered out, the attention module stacks all the features in-order to generate an (N, C) tensor. This tensor is padded by zeros to make it divisible by the number of desired points per-partition K, yielding the padded tensor $(\hat{N}, C)$. At last, the tensor is reshaped into a $(\frac{\hat{N}}{K}, K, C)$ tensor, so that classical attention is now applied to each of the $\frac{\hat{N}}{K}$ windows. Note that each of the windows are non-overlapping, meaning that the attention can be computed in parallel. Hence, the overall octree attention has a worst-case work of $O(\frac{\hat{N}}{K}k^2)$ and worst-case asymptotic span of $O(K^2)$. Since K is usually a small number (32 for example), this is a much more desirable overall performance.

Towards a laplacian label predicting network, we adapt the octformer as the backbone of our architecture. More precisely, we use the OctFormer network to generate a concatenation of L latent encodings, where L is the number of Octformer stages (contiguous blocks). Note that each of the latent encodings have a downsampled dimension, and needs to be upsampled to generate N features. Once upsampled, the features are fed to a Feature Pyramid Network (FPN) that converts the high dimensionality encoder channel width to the desirable output width, which in our case would be the number of possible laplacian labels. Note that this laplacian label prediction/segmentation architecture is identical to that for the task of semantic segmentation, with the only difference being the number of classes/labels at the final output.

3.4 Downsampling with SampleLapNet

Once SampleLapNet has been trained to segment points in a point cloud into Laplacian intervals, we freeze the learned weights. Then, we integrate the model into the semantic segmentation task pipeline to showcase its utility as a downsampling method for various downstream tasks.

An essential consideration when integrating SampleLapNet into any point cloud learning architecture is the downsampling criteria, specifically, how to select points based on Laplacian labels. To illustrate this, we can argue for at least two distinct types of sampling solely for the task of semantic segmentation:

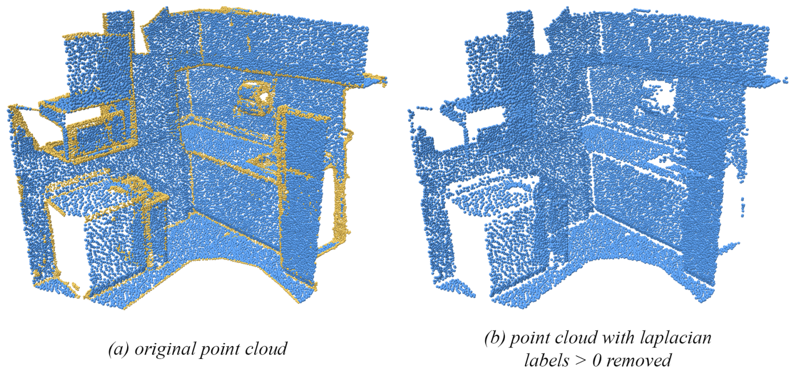


Figure 3.3: An example of downsampling by dropping $> l$ labelled points. Note that blue points have labels 0 and yellow points have labels ≥ 1

- **Drop all $> l$ labelled points**: This downsampling method entails discarding all points with a Laplacian label greater than a given threshold l, typically near the maximum importance label. Although this method doesn't significantly decrease the number of points, it efficiently divides the scene into patches of low-importance regions. Theoretically, this could enhance the accuracy of a semantic segmentation network by aiding in the distinction of boundaries between different segments.
- **Stochastic sampling with label-based probabilites**:"This downsampling method involves dropping a specified number of points p, randomly sampled from the point cloud. The sampling probability is inversely proportional to the numerical label of the sampled points. This probability distribution can be dynamically computed by applying a softmax function on the output of the Laplacian predictor. The rationale behind this downsampling method lies in the principle that points of lower importance are less likely to significantly affect the encoding of the point cloud, and therefore can be safely dropped.

Finally, it's crucial to consider whether SampleLapNet is integrated during training or inference. Our research primarily revolves around the tradeoff between two aspects. On one side of the coin is the aim to reduce the tradeoff between inference/training time and accuracy, while on the other side is the goal to diminish the tradeoff between accuracy and inference time. This tradeoff is seen more clearly when considering whether or not SampleLapNet is incorporated during training.

If integrated during training, SampleLapNet contributes to convergence by augmenting the training data and mitigating the risk of overfitting in the downstream model. While SampleLapNet is unlikely to significantly improve training times due to the substantial overhead of backpropagation, it also doesn't extend training time extensively. This is because the overhead of inferring Laplacian labels is comparatively smaller than training the downstream model.

Conversely, if SampleLapNet is exclusively integrated during inference, it has the potential to significantly enhance inference time, particularly when the ratio of the downstream model's size to the Laplacian predictor's size is large—a topic we'll look into into further in Chapter 5. The rationale behind this lies in the constant overhead of SampleLapNet, which often pales in comparison to the reduction in overhead achieved through downsampling. This is because inference time scales non-linearly with input size. However, in practice, applying SampleLapNet solely during inference presents a challenge related to the generalizability of the learned model. A downstream model lacking in generalizability may experience a notable reduction in accuracy due to such downsampling.

Given that semantic segmentation remains a challenging benchmark for point cloud learning, integrating our Laplacian predictor to downsample the point cloud during inference only could substantially compromise accuracy. Therefore, we opt to concentrate on the alternative side of the tradeoff: enhancing the training accuracy of semantic segmentation without imposing significant overhead on training.

3.5 Alternative Architectures

Towards our goal of learning the laplacian labels, we have explored alternative architectures to OctFormer. Below is a selected list of alternative experiments:

- **MLP-based Architecture (PointNet):** Our earliest attempt at training a laplacian estimator was using the foundational PointNet [4] model, a simple, MLP-based architecture that was the earliest in the field to apply learning on points directly. Our tests showed that PointNet was not suited to learn the laplacian. We reason that this is caused by the lack of local aggregation, as the architecture relies solely on a global feature vector to output per-point features.

- **Convolution-based Architecture (DGCNN):** Since the laplacian is a geometric local operation by its nature, one of our earlier work involved experimenting with a convolution based architecture, such that the learnt kernels become an estimate of actual laplacian computation. To test that intuition, we made use of Dynamic Graph CNN (DGCNN) [20], a benchmark architecture for point cloud convolution. We observed that such architecture, and similar CNN based were too slow for our purposes, since k-nearest neighbours (kNN) needed to be computed for each convolution operation, which became a huge overload as channel width increased.

- **Alternative Transformer Architecture (Point Transformer):** Before settling on OctFormer as the base architecture, we attempted to use a standard Point transformer [26] to perform laplacian label prediction. While the results were promising, we ended up opting for an Octformer architecture due to its better asymptotic performance with large input point clouds, something a Point transformer would struggle with.

Chapter 4

Experimental settings

This chapter details the different experiments that have guided our decisions with respect to training hyperparameters, task selection, and evaluation metrics.

4.1 Downstream Tasks & Datasets

Deep Learning models that learn on point clouds are usually assessed on a set of common benchmark tasks. These tasks are often applicable to real-world scenarios, and pose significant challenges to current models.

In our literature review, we found that the tasks of classification, semantic segmentation, instance segmentation and object detection to be recurring in point cloud learning research papers. From these four tasks, we filtered out semantic segmentation as the task of choice due to the reasons listed below:

- **Non-saturated State-of-the-art:** Unlike the task of classification, whose state of the art has surpassed 95% accuracy [?], semantic segmentation remains a challenging task to even state-of-the-art models, with current top results for the mean Intersection-over-Union (mIOU) being 0.75 for SemanticKITTI and only 0.39 for ScanNet200 [22]
- **Straightforward End-to-End training:** Unlike the task like object detection, which itself might involve semantic segmentation [17], semantic segmentation is a straightforward feature generation task. Input point clouds are fed as $N \times 3$ tensors, and the output is given by $N \times k$ tensor, where k is the number of labels, such that the features of each point represent the likelihood probabilities for that point to be of a particular label.
- **Rich annontated datasets:** Semantic segmentation has relatively richer datasets relative to other tasks. Such benchmark datasets in-

clude: SemanticKITTI [2], ScanNet20, ScanNet200 [5], and S3DIS [1] to name a few.

Additionally, we have chosen to use ScanNet as the benchmark dataset of choice due to its large collection of real-world scans (> 1500 real-life indoor scenes), and its two benchmark options, namely 20 classes and 200 classes, the later of which remains extremely challenging to state-of-the-art models as stated earlier.

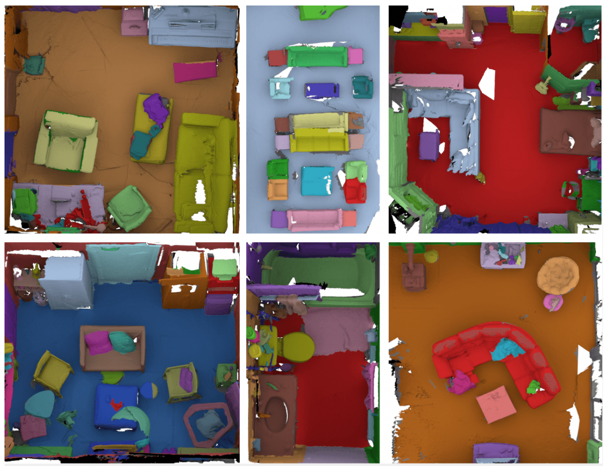


Figure 4.1: Example scenes from the ScanNet dataset. Note that different colors indicate different object class. In our experiments, we extract only the points to form point clouds from given meshes

4.2 Laplacian Labels

So far, we have built the idea of estimating the importance of points in a point cloud based on the interval within which they in the laplacian distribution. However, the way we define such intervals has a great influence on the convergence of our laplacian predictor. In this section, we briefly discuss the experimental choices pertaining laplacian labels.

Firstly, we need to decide on the metric that we are using to generate the labels. The laplacian of the identity function $f(x, y, z) = (x, y, z)$ is not a scalar value, and is indeed a vector that describes the average deviation of a point in all three coordinates. There are various ways to aggregate this information into a scalar value, but we choose the simplest and most effective way to approach that, which is simply to compute the norms of the laplacian vectors, that is compute $||\Delta f||_2$. The reason for this choice is that it limits high importance points to truly divergent points (corners, edges), and is that it combines the directional Laplacian in a non-negative fashion, preventing the values from cancelling out.

Throughout our research, we've noted that points within a point cloud often exhibit a Laplacian norm distribution resembling a Boltzmann-Maxwell distribution. This distribution typically entails numerous points clustered around the minimum Laplacian norms, with fewer points possessing above-average norms. Figure 4.2 depicts this characteristic. Given the positively skewed nature of this distribution, the mean surpasses the median, indicating that the majority of points in any given point cloud have below-average Laplacian norms. Additionally, there exists a range of points with above-median Laplacian norms but still below the mean due to the skewed distribution. These observations suggest that partitioning the Laplacian norm distribution into equal-width intervals would result in the initial intervals containing a significant number of instances, while the tail intervals would contain very few.

Towards a more reasonable partitioning, we employ the following scheme to produce the intervals:

1. Compute all $100(1 - \frac{i}{k})$ percentiles for $0 < i < k$
2. Compute the upper quartile of the laplacian norm, and redistribute the points into two intervals L_1 and L_2 corresponding to points below and above upper quartile.
3. for all the points in L_2, further partition the interval into $\frac{k}{4}$ partitions, according to the percentiles computed earlier.
4. Label the points in order of interval, with lowest interval having label 0, and largest interval having label $\frac{k}{4}$, for a total of $1 + \frac{k}{4}$ interval.

The rationale behind this approach is twofold: firstly, we aim to assign the lowest importance label to points with below-average Laplacian norms. Secondly, we strive to maintain a consistent distribution of points across various training instances to facilitate model convergence. By utilizing the upper quartile—excluding extreme distributions, where it may not align with the mean—we achieve both objectives simultaneously.

In the remaining distribution, we aim to establish clearer distinctions beyond a single label. As we move further to the right, the number of points decreases, yet their geometric significance increases. However, we also strive to avoid creating an excessive number of intervals that could render the partitioning meaningless. This delicate balance is precisely captured by the hyperparameter k: larger values of k result in potentially numerous but vague geometric partitions, while smaller values yield fewer yet more distinct distributions. We selected $k = 12$, providing three partitions, which we believe strikes a suitable balance between both considerations.

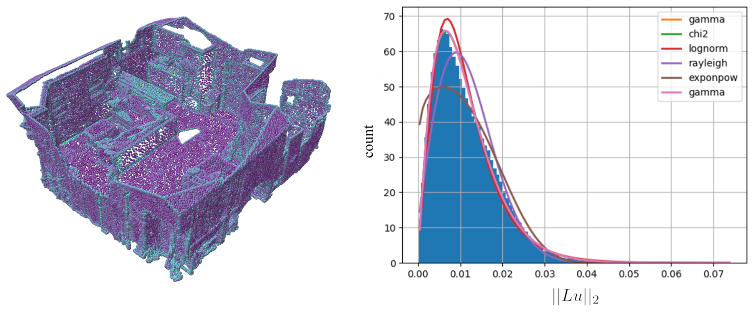


Figure 4.2: Picture (left) shows the laplacian norm distribution over a scene point cloud with cyan hues indicating larger laplacian norms. Plot (right) depicts the skewed head distribution of the laplacian norms in a scene point cloud

4.3 Training Loss Function

The nature of the laplacian predictor, at its core, is a per-point classification task, and hence it is very natural to utilize a loss function that is commonly used to train classification neural networks. In our experiments, we use the cross-entropy loss, which for a predicted discrete probability distribution of classes $\boldsymbol{x} = (x_1, \cdots, x_C)$, and a truth target $y \in 1, \cdots C$, is given by the following equation:

$$CrossEntropy(\boldsymbol{x}, y) = -\sum_{c=1}^{C} y \log(x_c)$$

Since the output of our final layer might not necessarily be a probability distribution, we apply a softmax function to normalize the values. That is, if the final layer output is vector $\boldsymbol{z} = (z_1, \cdots, z_C)$, we can compute $\boldsymbol{x}$ using the formula below:

$$\boldsymbol{x} = (x_1, \cdots, x_c) \quad x_i = \frac{\exp(z_i)}{\sum_{c=1}^{C} \exp(z_c)}$$

The cross-entropy losses are computed per point and aggregated (compute the average loss), which is then used to back propagate through the network.

Nevertheless, applying the base cross-entropy loss to our logits is not ideal. As we discussed in the previous section, we have a distribution of classes where 75% of all points are labelled as class 1. This implies that the model is more likely to converge to a state that satisfies the labelling of the dominant target, and the mis-classification of the other labels has lesser impact on the back propagation of gradients. To fix this, we assign weights to the classes that's inversely proportional to their distribution. Hence, we assign $w_1 = \frac{4}{3}$ to class 1, and $w_i = 12$ for the rest three. The overall modified cross entropy formula is shown below:

$$CrossEntropy(\boldsymbol{x}, y) = -\sum_{c=1}^{C} y \cdot w_c \log(x_c)$$

Note that because the semantic segmentation task is also a per-point classification task, we also use the cross-entropy loss function to train the semantic segmentation network. However, we keep the default weights because we are not informed on the distribution of semantic classes.

4.4 Evaluation Metrics

There are three main evaluation metrics that are relevant to our Laplacian predictor. In this section, we provide a brief mathematical background on these metrics:

- **Accuracy**: the accuracy of a single point cloud instance is simply the percentage of points that the laplacian predictor/semantic segmentator has predicted correctly. Accuracy provides a broad view of the performance of a model, but is not a good indicator overall due to the imbalanced nature of most instances.
- **Recall**: For our laplacian predictor, it is important to know the percentage of high-importance points that are classified correctly, that is, the ratio of true-positive instances, to true positive and false negatives instances $\frac{TP}{TP+FN}$. We consider all points that are not the lowest class positive, and all that are at the lowest class negative, and compute the recall. Note that we are not interested in computing the precision for two reasons: (1) because the upper quartile could be below the mean, some points that end up having truth labels as low-importance should more ideally be high importance, and (2) because we care more about the completeness of our predictor over its soundness.
- **mean Intersection over Union (mIoU)**: Intersection over union is a measure of the accuracy of "alignment" of two sets. It is simply the ratio of the size of the set of intersection over the size of the set of union. In our case, we compute the IoU of the sets corresponding to each of the classes. That is, we create the sets $X_1, \cdots X_C$ from the predictions and $Y_1 \cdots Y_C$ from the ground truth, and compute for each of the C classes the IoU $\frac{|Y_i \cap X_i|}{|Y_i \cup X_i|}$, those individual IoU's are averaged to obtain mIoU.

Chapter 5

Results

In this section, we detail the results of training the different components of our architecture, and the results of various experiments that were carried out to find out the optimal sampling strategy.

5.1 Laplacian Segmentation (Prediction)

As detailed in the previous section, our architecture relies on the a laplacian prediction network that predicts the relative magnitude (interval) of the laplacian norm, and hence the geometric importance of each point. We pre-train this network before incorporating it into the full segmentation network.

Channel width	96
FPN channel width	168
Block Numbers	2, 2, 6/18*, 2
Attention heads	6, 12, 24, 2
Learning rate	1.5×10^{-3}
Optimizer	AdamW with $\lambda = 0.05$
LR Scheduler	Step Warmup

Figure 5.1: Summary of the architecture and training hyperparameters for laplacian segmentation. Note that the third stage has 18 blocks for the base OctFormer and 6 for the small variant

Training settings We train the OctSegFormer (OctFormer + FPN head) architecture discussed in chapter 3. In order to evaluate the generalizability of laplacian segmentation models, we test two different network sizes, namely octsegformer-small and octsegformer. Figure 5.1 show a brief summary of the training hyperparameters.

Network	IoU(X_1)	IoU(X_1^c)	mIoU
OctSegFormer-Small	**65.1%**	15.4%	28.4%
OctSegFormer	63.9%	**15.7%**	**28.7%**

Figure 5.2: Mean Intersection over Union (mIOU) for Laplacian segmentation on the ScanNet validation dataset with two different model sizes.

Network	Recall	Accuracy
OctSegFormer-Small	91.5%	**60.8%**
OctSegFormer	**91.6%**	60.5%

Figure 5.3: Accuracy and Recall for Laplacian segmentation on the ScanNet validation dataset with two different model sizes.

Results We analyze the two variants of OctSegFormer in Figure 5.2 and 5.3, focusing on Recall, Accuracy, and mIoU. Despite observing a very small mIoU in both cases, which could imply poor performance, it's crucial to recognize that mIoU is heavily influenced by set sizes. The smaller the ground truth set, the more significant the impact of misclassification becomes. To delve deeper into this phenomenon, we compute the IoU of X_1, representing the set of all low-importance points. Results show that we achieve acceptable outcomes in both scenarios, with 63.9% for the base model and 65.1% for the smaller model.

Regarding accuracy and recall, it's notable that the accuracy in both models is relatively low. However, as discussed in Section 4.3, our model tends to 'overshoot' when labeling, which is preferable to missing important points entirely. This is evidenced by our recall values, both exceeding 90%. Thus, we can infer that the models effectively classify important points. Additionally, considering all evaluation metrics, we observe that a smaller model is adequate for learning Laplacian labels, with negligible improvements from the larger model. This suggests that further optimization for model size is feasible, consequently reducing the overhead when integrated with a downstream task network.

To underscore the advantage of our Laplacian prediction network, we compare its inference times with the average time required to compute the mesh Laplacian for a point cloud. We conduct this comparison using the validation dataset from ScanNet. Figure 5.5 illustrates the results. Notably, OctSegFormer operates on average at over four times the speed of the mesh laplacian algorithm, and almost 16 times the speed of the point cloud laplacian algorithm. This demonstrates the unnecessary complexity of computing the exact Laplacian solely for labeling points based on their importance.

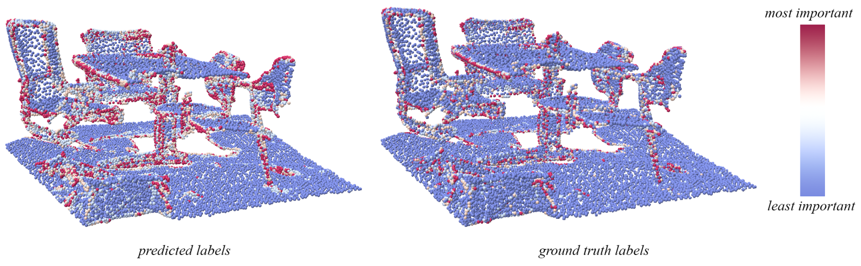


Figure 5.4: Example result on a point cloud from scannet dataset

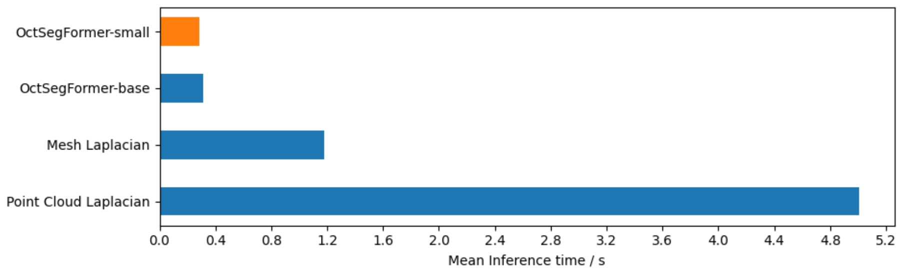


Figure 5.5: Comparison of Inference time of SampleLapNet's laplacian predictor with laplacian algorithms

5.2 Semantic Segmentation

Setup Prior to integrating our Laplacian prediction network, we pre-train a base OctSegFormer for semantic segmentation using the ScanNet dataset. The hyperparameters employed in this run mirror those depicted in Figure 5.1. The sole difference lies in the FPN output channel width, set to 20 to accommodate the generation of probabilities for the 20 classes in the standard ScanNet semantic segmentation task. Figure [insert figure number] displays the training plots for this phase.

Following training, we initiate two runs to assess the impact of the Laplacian predictor on learning. The first run incorporates standard point cloud augmentation techniques, including random flip, rotation, and jitter. In the second run, we employ the aforementioned augmentation methods alongside point cloud downsampling with a probability of $p = 0.2$ per instance. We utilize the $> l$ downsampling technique outlined in Section 3.3, with $l = 1$, implying the retention of only the low-importance points (as in Figure 3.3).

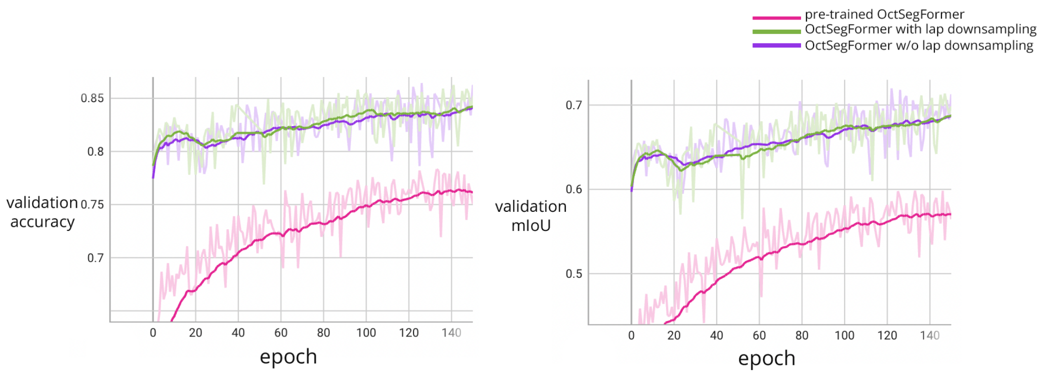


Figure 5.6: Comparison of validation mIoU and accuracy over epochs for the three runs. It's important to mention that we initialize the green and purple runs with pre-trained model weights.

Network	val. mIOU	val. mAcc	training time / s
OctSegFormer-base	60.2%	60.2%	**0.340**
OctSegFormer-augment	72.0%	86.4%	0.430
OctSegFormer-laplace	**72.9%**	**87.1%**	0.431

Figure 5.7: Semantic Segmentation performance on ScanNet20 with and without Laplacian downsampling, along with and without augmentation. It's worth noting that both augmentation and Laplacian downsampling include the standard random augmentations mentioned earlier.

Results We compare the results of both runs, alongside the original unaugmented run, in Figures 5.6 and 5.7. It's notable that the base augmentations significantly enhance the model's performance, as evident from the plots. However, these augmentations do incur increased training time per batch, as shown in the table above.

Regarding Laplacian downsampling, while there is an improvement in both mIoU and Accuracy, the enhancement is relatively modest. One could argue that this downsampling indeed contributes to improved training, but it's challenging to ascertain whether this is due to the effectiveness of Laplacian downsampling itself or simply because we are randomly removing some points from the scene, akin to random cropping. Nonetheless, it's worth mentioning that the model trains in nearly the same time per batch as its counterpart, providing some assurance of potential marginal improvements without added training overhead.

Chapter 6

Conclusions & Future Work

6.1 Discussion

This thesis presents SampleLapNet, a novel approach to downsampling that combines both geometric intuitions as well as learning-based methods to implement an efficient yet geometry-aware downsampling method on point clouds. We believe this work is the first of its kind in the domain of point cloud downsampling. By performing "smarter" downsampling, we provide downstream models more leverage to incorporate deeper and more generalizable networks without inference time concerns. In addition, we foresee it application in processing point cloud sequences, in particular when to comes to real-time inference.

Despite this, we recognize certain limitations in our downsampling network. We present a few of these as follow:

- **Diminishing returns:** As we've observed, enhancements in inference times and/or accuracy are frequently limited by the prevailing generalizability constraints inherent in point cloud deep learning models. Attempting more aggressive downsampling, particularly during inference only, yields inferior results due to these constraints, even when geometrically insignificant points are sampled out. Further investigation is needed to understand why this phenomenon occurs.
- **Supervised learning:** Recent work in the domain of point cloud learning has seen significant incorporation of unsupervised and self-supervised learning models. Such models address the challenges of limited access to labelled data. Nevertheless, we have adopted a supervised learning model in this project, which limits our work to datasets with cleaned datasets that have computable laplacians.

- **One-time Downsampling:** Our downsampling method provides a very intuitive geometric correlation with the idea of variability as a measure of importance. Despite that, our downsampling is limited to three-dimensional measure of laplacian, as our current method only learns the laplacian labels and not the matrix itself. Downstream layers with more than 3 channels cannot benefit from additional laplacian-based downsampling, and it is not clear whether the laplacian values provide a notion of importance when dimensionality exceeds three.

6.2 Future Work

Due to the novelty of the concept, our laplacian-based downsampling model has promising future prospects that can be explored in future research. We list a few of these tangents below:

- **Laplacian beyond labels:** Although our primary goal in predicting the Laplacian labels is to facilitate geometry-aware downsampling, we contend that an accurate and efficient Laplacian estimator, which directly aligns the model with a Laplacian function instead of simply labels, holds broader applications beyond downsampling. These applications include real-time physics inference and modeling, as well as real-time point cloud denoising through mean curvature flow.

- **Downsampling ensemble:** One of the important aspects that distinguish our downsampling methodology is its independence from the downstream task. Nevertheless, it might be desirable to train downsampling models that are fine tuned for a particular task. A potential middle-ground between those two approaches is an ensemble combination of both methods, that scores point importance as a weighted (and potentially learnt) combination of both scoring methods. We reason this could combat overfitting in task-based downsampling methods, and improve overall task accuracy.

Bibliography

[1] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, Feb. 2017.

[2] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[3] A. Boulch. Convpoint: Continuous convolutions for point cloud processing. *Computers Graphics*, 88:24–34, 2020.

[4] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[5] 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.

[6] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4338–4364, 2021.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[8] M. D. P. S. Keenan Crane, Fernando de Goes. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 courses*, SIGGRAPH '13, New York, NY, USA, 2013. ACM.

[9] I. Lang, A. Manor, and S. Avidan. Samplenet: Differentiable point cloud sampling. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7575–7585, 2020.

[10] J. Lee, M. Sung, H. Kim, and T.-K. Kim. Pop-out motion: 3d-aware image deformation via learning the shape laplacian. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18532–18541, June 2022.

[11] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. Pointcnn: Convolution on x-transformed points. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[12] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5105–5114, Red Hook, NY, USA, 2017. Curran Associates Inc.

[13] N. Sharp and K. Crane. A Laplacian for Nonmanifold Triangle Meshes. *Computer Graphics Forum (SGP)*, 39(5), 2020.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[15] X. Sun, Z. Lian, and J. Xiao. Srinet: Learning strictly rotation-invariant representations for point cloud classification and segmentation. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 980–988, New York, NY, USA, 2019. Association for Computing Machinery.

[16] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6410–6419, 2019.

[17] O. Unal, L. Van Gool, and D. Dai. Improving point cloud semantic segmentation by learning 3d object detection. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 2949–2958, 2021.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[19] P.-S. Wang. Octformer: Octree-based transformers for 3d point clouds. *ACM Trans. Graph.*, 42(4), jul 2023.

[20] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019.

[21] C. Wu, J. Zheng, J. Pfrommer, and J. Beyerer. Attention-based point cloud edge sampling. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5333–5343, Los Alamitos, CA, USA, jun 2023. IEEE Computer Society.

[22] X. Wu, L. Jiang, P.-S. Wang, Z. Liu, X. Liu, Y. Qiao, W. Ouyang, T. He, and H. Zhao. Point transformer v3: Simpler, faster, stronger, 2024.

[23] C. Zeng, W. Wang, A. Nguyen, J. Xiao, and Y. Yue. Self-supervised learning for point cloud data: A survey. *Expert Systems with Applications*, 237:121354, 2024.

[24] J. Zhang, X. Lin, and X. Ning. Svm-based classification of segmented airborne lidar point clouds in urban areas. *Remote Sensing*, 5(8):3749–3775, 2013.

[25] H. Zhao, L. Jiang, J. Jia, P. Torr, and V. Koltun. Point transformer. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16239–16248, 2021.

[26] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun. Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16259–16268, October 2021.