Approach to Deep Halo using Graph Neural Networks

Aritra Mukhopadhyay
National Institute of Science Education and Research
Bhubaneswar, Odisha 751005, India
4th year, Integrated M.Sc. Physics
Roll No.: 2011030

(Dated: April 23, 2024)

The problem we address in this work is substructure-finding in dark matter halos from gravity-only N-body simulations, which is typically solved using percolation algorithms that require simulation history and increase the simulation cost. To overcome these limitations, we apply point-cloud segmentation techniques using Graph Neural Network (GNN) based methods.

Specifically, given a halo represented as a point cloud, our goal is to classify each point into either belonging to a halo or the background, and if it belongs to a halo, differentiate which halo it is a part of. To address the challenges of large datasets and significant class imbalance between the background and sub-halo classes, we implement techniques for efficient processing of large datasets and methods to balance the class distribution.

Our approach uses point cloud data to train a neural network that can classify points in these simulations without requiring simulation history. This results in accurate and efficient substructure identification.

I. INTRODUCTION

Substructure-finding in dark matter halos from gravity-only N-body simulations is an important problem in astrophysics and cosmology. These simulations generate large datasets that represent the distribution of dark matter particles in the universe, which are used to study the formation and evolution of galaxies and other structures. The substructure-finding problem involves identifying and classifying halos within these simulations based on their density and spatial distribution.

Traditionally, percolation algorithms have been used to solve this problem by connecting neighbouring particles in a simulation to form clusters or halos. However, these methods require simulation history and increase the simulation cost due to the need for multiple time steps. We apply point-cloud segmentation techniques using Graph Neural Network (GNN) based methods to address these limitations.

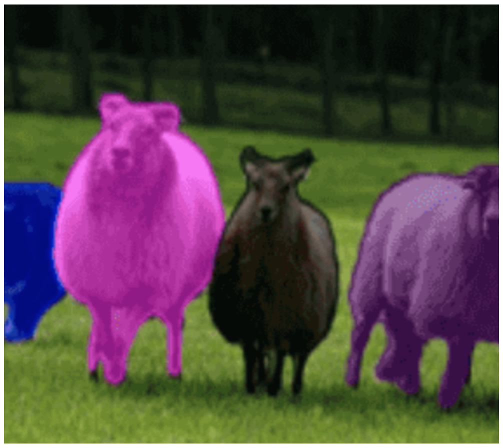
FIG. 1: Instance Segmentation Example from [1]: here we can see in this image how the pixels containing the sheep are differentiated from the background and also each sheep is coloured with different hues. Thus the model not only classifies each pixel into one of the classes, but also tells which sheep that pixel belongs to.

II. BACKGROUND

This section provides a brief overview of the technologies and techniques used in our approach. We discuss the instance segmentation task on images, the DynamicEdgeConv Graph Neural Network, Octree algorithm for reducing points, and the SMOTE technique to address class imbalance. Although we explored Siamese networks, they were not included in the final solution.

A. Instance Segmentation task on images, and how it is analogous to our problem

Instance segmentation is a computer vision task that deals with detecting, classifying, and segmenting individual objects within an image. This technique assigns unique labels to each object instance in the scene, enabling pixel-wise mask generation for each distinct object. In other words, instance segmentation not only identifies the classes of objects present in an image but also distinguishes between different instances of the same class as shown in Figure 1.

In our problem, we focus on substructure-finding in dark matter halos from gravity-only N-body simulations using point cloud segmentation techniques based on Graph Neural Network (GNN) methods. Here, each particle can be either labeled as background (meaning it is not a part of any subhalo) or classified as a part of one of the subhalos.

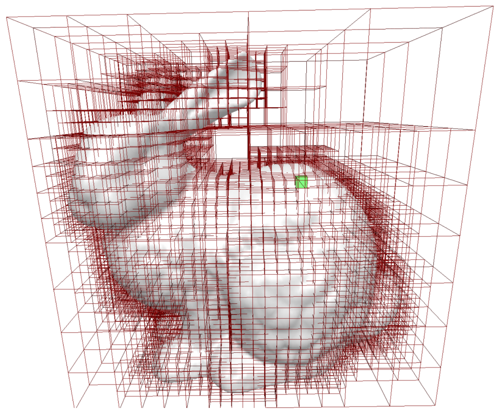

FIG. 2: Octree Representation of a model of a rabbit taken from [2]. Here we can see how the voxels become smaller and smaller as it reaches closer to the surface of the model, and becomes large away from the model surface

The instance segmentation task on images and our point cloud classification problem share similarities in their goals: both tasks aim to classify individual elements within their respective data types (pixels for images, particles for point clouds) into distinct classes and instances. By applying instance segmentation concepts from image processing, we can identify and separate individual subhalos within the point cloud before classifying them, potentially improving classification performance by allowing for more focused analysis of each object in the scene.

Instance segmentation on images involves classifying and differentiating between instances of objects at the pixel level. In our problem, we apply similar concepts to point clouds, where particles are classified into distinct classes and instances, enabling improved substructure-finding within dark matter halos from gravity-only N-body simulations.0

B. Octree

The Octree algorithm is a hierarchical spatial partitioning technique used to efficiently manage and process large datasets, particularly those with varying densities. In our case, we aim to reduce the number of points in our point cloud data, which will help optimize both time and memory usage.

Initially, we could have divided the space into grids, treating all points inside a single cube (voxel) as a single point. This approach would have introduced a new features for each voxel: mass, equal to the number of points within that voxel. We also recalculate the position and the velocities, as the mean of all points in that voxel.

However, our dataset presents a challenge due to its varying density. Some regions are extremely dense, while others are extremely sparse. A naive grid-based approach would be wasteful, as many voxels would remain empty, while others would contain an overwhelming number of points. To address this issue, we employ the Octree algorithm. We can see in Figure 2 how we use different sizes of voxels all over the grid.

The Octree algorithm is a recursive partitioning method that adaptively subdivides the 3D space into smaller regions based on the density of points within each region. This approach ensures that dense regions are divided into smaller voxels, while sparse regions are covered by larger voxels, thereby optimizing memory and computational resources.

Here's a step-by-step explanation of the Octree algorithm:

1. **Root Node Creation:** The algorithm begins with the creation of a root node, which represents the entire 3D space.

2. **Point Insertion:** Each point in the dataset is inserted into the Octree by recursively traversing the tree. At each node, the point is directed to one of eight possible child nodes based on its spatial location (e.g., top-left-front, bottom-right-rear, etc.).

3. **Node Splitting:** When a node exceeds a predetermined capacity (i.e., the maximum number of points it can hold), it is split into eight smaller child nodes. This process continues recursively until each node meets the capacity constraint or the minimum voxel dimensions is reached.

4. **Voxel Creation:** Once all points have been inserted, each leaf node in the Octree represents a voxel. The mass and position features are calculated for each voxel as described earlier.

This algorithm doesn't just help in reducing the number of points, the tree structure prepared in the process helps in searching nearest neighbours in the whole dataset really fast. Initially given a point, searching for the k nearest neighbours used to take $O(n)$ time complexity; but low it reduced to $O(log(n))$ time which is much better. This is really helpful because we need to find this k nearest neighbours multiple times along the process; once while doing smote and also while running the GNN network.

C. Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are a class of neural networks designed to handle graph-structured data. They operate by iteratively applying message passing, aggregation, and update operations on each node in the graph. At each iteration, or layer, GNNs update the features of each node based on its neighbors' features.

To illustrate how GNNs work, let's consider a simple example of a graph with some nodes and edges where each node has an initial feature vector. The message passing step involves computing a message for each edge in the graph based on the features of its endpoints. For instance, if we use a simple linear function to compute the messages, then the message from node A to node B would be:

$$m_{A \to B} = W \cdot h_A + b \tag{1}$$

where h_A is the feature vector of node A, and W and b are learnable parameters. The message from node B to node A would be computed similarly.

Once the messages are computed for all edges in the graph, they are aggregated at each node using a permutation invariant function such as sum or mean. For instance, if we use sum as the aggregation function, then the aggregated message at node B would be:

$$a_B = \sum_{i}^{\text{neighbours of B}} m_{i \to B} \tag{2}$$

Finally, the updated features of each node are computed based on its aggregated messages and its previous features using a learnable function such as a linear layer or a multi-layer perceptron (MLP). For instance, if we use an MLP to update the features of node B, then:

$$h'_B = \text{MLP}([h_B, a_B]) \tag{3}$$

where h'_B is the updated feature vector of node B.

The message passing, aggregation, and update operations are repeated for multiple layers in GNNs, allowing them to capture increasingly complex patterns in the data. In our approach, we use a specific type of GNN called Dynamic Edge Convolution (DynamicEdgeConv), which is well-suited for point cloud data.

D. Dynamic Edge Conv (A Specialized GNN)

Our approach uses Dynamic Graph CNNs [3] as the Graph Neural Network (GNN) to classify points in a halo. DynamicEdgeConv is a type of GNN that can handle varying graph structures and dynamically update edge features based on node features. This makes it well-suited for point cloud data, where the number of nodes and edges can vary significantly between halos.

DynamicEdgeConv works by first computing the k-nearest neighbors (k-NN) for each point in a halo using the k-NN algorithm. The resulting nearest neighbor information is used to construct an edge list that defines the graph structure. Specifically, for each point, its k-NN are considered as its edges. Th graph thus created is then passed through the GNN to get the model predictions.

The updated node and edge features are then used as input to a classification layer that predicts whether each point belongs to a halo or the background. The DynamicEdgeConv module is applied multiple times in our approach, allowing the model to capture increasingly complex patterns in the data.

DynamicEdgeConv has several advantages over other GNN architectures for point cloud data. First, it can handle varying graph structures and dynamically update edge features based on node features, making it well-suited for point cloud data. Second, it is computationally efficient, as it only computes the k-NN using faster algorithms which we would talk about later. Finally, DynamicEdgeConv has been shown to achieve state-of-the-art performance on several benchmark datasets for point cloud classification.

E. Siamese Networks

A Siamese Network [4] consists of two or more identical subnetworks, where each subnetwork processes different inputs while sharing the same weights. This architecture is used to learn an embedding space where similar items have vector embeddings close together and dissimilar items have vector embeddings farther apart.

To train a Siamese network for point cloud classification, we need to define a distance metric in the embedding space. Commonly used metrics include Euclidean distance and cosine distance:

1. **Euclidean Distance:** This is the most common distance metric, measuring the straight-line distance between two points in an embedding space. Mathematically, it is defined as the square root of the sum of squared differences between corresponding dimensions in two vectors. For example, given two vector embeddings, $\vec{A}$ and $\vec{B}$, their Euclidean distance would be $d(\vec{A},\vec{B}) = \sqrt{\sum(\vec{A}-\vec{B})^2}$

2. **Cosine Distance:** This metric measures the cosine of the angle between two vectors in an embedding space. It ranges from -1 to 1, where 0 indicates orthogonal vectors (i.e., dissimilar), and 1 or -1 indicates identical vectors (i.e., similar). Cosine distance between two vectors $\vec{A}$ and $\vec{B}$ is defined as $d(\vec{A},\vec{B}) = 1 - \frac{\vec{A}\cdot\vec{B}}{|\vec{A}||\vec{B}|}$

In addition to Euclidean and cosine distances, other notions of distances can also be used depending on the specific problem requirements.

To train a Siamese network, we use a loss function called **contrastive loss**. This loss function encourages similar items (i.e., anchor-positive pairs) to have vector embeddings close together and dissimilar items (i.e., anchor-negative pairs) to have vector embeddings farther apart in the embedding space.

Mathematically, contrastive loss is defined as:

$$L(y,d) = y \times (d^2) + (1-y) \times max(margin - d, 0)^2$$

where y is a binary variable indicating whether the input pair is similar (1) or dissimilar (0), and d is the distance between their vector embeddings. The margin hyperparameter determines how far apart dissimilar items should be in the embedding space.

For example, consider training a Siamese network with cats and dogs as input pairs. If we want to learn an embedding space where cats and dogs are separated, we would define anchor-positive pairs as two images of the same cat or dog, and anchor-negative pairs as two images of different animals (i.e., a cat and a dog). By minimizing contrastive loss during training, the Siamese network will learn an embedding space where cats and dogs have distinct vector embeddings, making it easier to distinguish between them.

In our point cloud classification problem, we can use a similar approach by defining anchor-positive pairs as two particles from the same subhalo or background and anchor-negative pairs as particles from different subhalos or background. By minimizing contrastive loss during training, the Siamese network will learn an embedding space where vector embeddings of particles from the same subhalo or background are close together, while those from different subhalos or background are farther apart, improving substructure-finding within dark matter halos from gravity-only N-body simulations.

We used this initially; we had learned about it but did not use it at the end. Other naive methods, like simple binary classification, proved to be equally good in this case.

F. SMOTE: Synthetic Minority Over-sampling Technique

In many real-world datasets, especially in classification problems, the distribution of classes is often imbalanced, meaning that one class significantly outnumbers the others. This can lead to biased models that perform poorly on minority classes since they have less representation in the data.

SMOTE, which stands for Synthetic Minority Oversampling Technique [7], is a method used to address imbalanced datasets by generating synthetic samples for the minority class. By doing so, it aims to balance the class distribution and improve the performance of classifiers, particularly in scenarios where the minority class is underrepresented.

The SMOTE algorithm: SMOTE works by creating synthetic instances of minority class samples. Here's a step-by-step explanation of the algorithm:

1. **Selecting a Minority Instance:** SMOTE starts by randomly selecting a minority instance from the dataset.

2. **Finding k-Nearest Neighbors:** Once a minority instance is chosen, SMOTE identifies its k nearest neighbors in the feature space. The value of k is a parameter set by the user.

3. **Generating Synthetic Samples:** SMOTE generates synthetic samples by interpolating between the chosen minority instance and its nearest neighbours. For each feature, SMOTE randomly selects a value within the range defined by the feature values of the instance and its neighbours.

4. **Adding Synthetic Samples:** The newly generated synthetic samples are added to the dataset, effectively increasing the representation of the minority class.

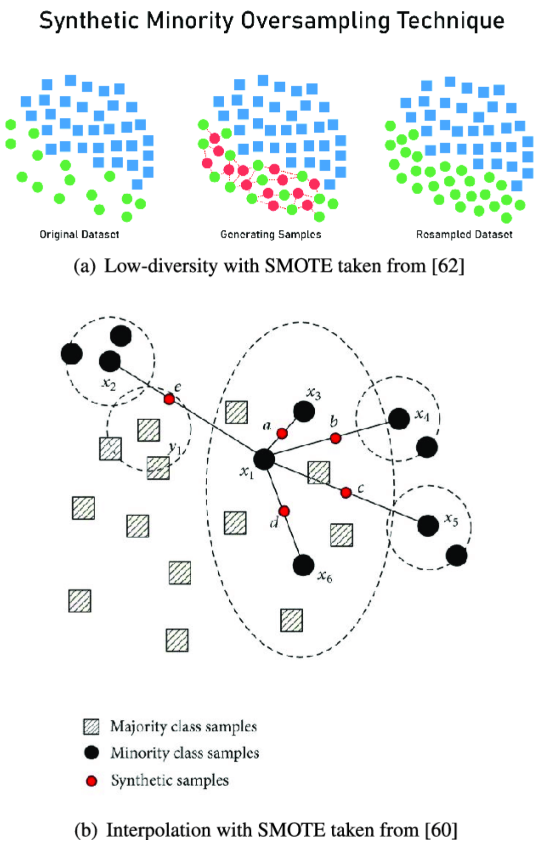

FIG. 3: Smote Illustration taken from [5, 6]

By repeating this process for multiple minority instances, SMOTE helps to balance the class distribution and mitigate the issues associated with imbalanced data.

III. EXPERIMENTS

In this section, we describe the experimental setup and results of our approach. We focus on two key components: (1) the efficient representation of point cloud data using an Octree structure, which enables significant reduction in the number of points while preserving essential information; and (2) the application of Graph Neural

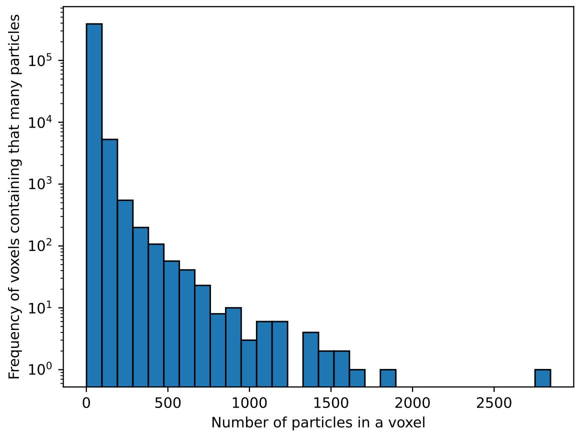

FIG. 4: Log scale frequency plot of masses of the voxels (mass is equivalent to number of particles in that voxel)

Networks (GNNs) for classification tasks on this reduced dataset. We provide a detailed explanation of our experimental methodology, including the minute details of our implementation, and present the results obtained from these experiments.

A. Point Reduction with Octree

We worked with the smallest data available on the site for now. The data is available on the website `https://www.tng-project.org/data/`. There we took the gravity-only data of the lowest resolution and worked on it. The idea was to develop the algorithm and codes for the lowest one first and then try the bigger data later. So we worked with the data named `TNG50-4-Dark`. It's simulation volume is 51.7 mega parsec cube and has 270^3 ($\approx 2 \times 10^7$) particles.

The first problem was to reduce the number of particles by voxelization. As explained in II B we used the octree algorithm. We initially had 2×10^7. There are two tunable parameters in the Octree algorithm:

- `max_particles_per_voxel`: This is the upper limit of particles in a voxel. No voxel in principle should carry more than these many points. **We kept this value to 100.**

- `min_voxel_size`: Because this is a recursive algorithm, and that we knew that there are certain locations in the space where the density of particles is tremendously large. So we kept this check that if the voxel size reduces below this point, we would not divide the voxels any further even if it had far more particles than `max_particles_per_voxel`. **We kept this value to be 0.001.** This means, the minimum voxel size is 0.001 times the simulation length (which is 51.7 mega parsec here).

Upon converting the point cloud to octree these are the statistics of the data:

- Total number of voxels: 395286 ($\approx$ 2% of 2×10^7)

- Number of voxels with less than 100 particles: 388990 (98.4%)

- Number of voxels with more than 100 particles: 6292 (1.6%)

Figure 4 shows the log-scaled distribution of a number of particles in voxels.

B. Prediction Using GNN

In Figure 5 he have shown a halo at random. In this halo, we can see that if the background points are somehow removed, it would be really easy to classify the halo points into different halos using naive Machine learning algorithms like **DB-Scan**. So if we can train a model which can differentiate between the background and the foreground (semantic segmentation task), we can then run clustering algorithms on the foreground to do the instance segmentation task. So the tough problem of instance segmentation got boiled down to classifying the points into background and foregorund.

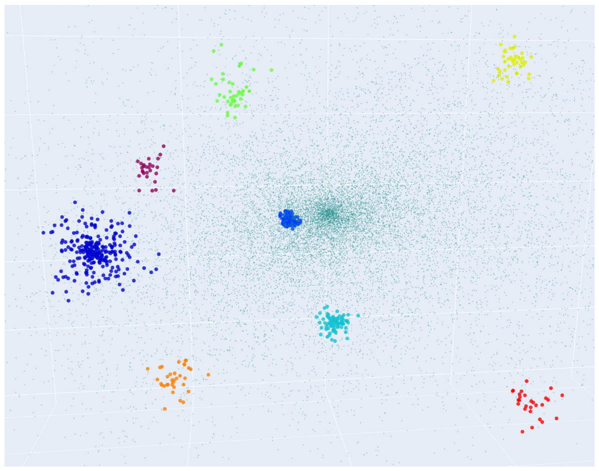

FIG. 5: Visualizing a halo: The different subhalos have been given different colours, and the background points have been made fine and given dark green colour

The proposed model consists of a Graph Neural Network (GNN) backbone, which leverages a simple Multi-Layer Perceptron (MLP) as its message passing function. This MLP, denoted by *MyNetwork*, is composed of three fully connected layers with ReLU non-linearity in between, specifically $7\times2 \to 8 \to 8 \to 8$. The GNN backbone employs this MLP to update node features during the message passing step, as described in Equation 1. The output from the GNN backbone is then passed through

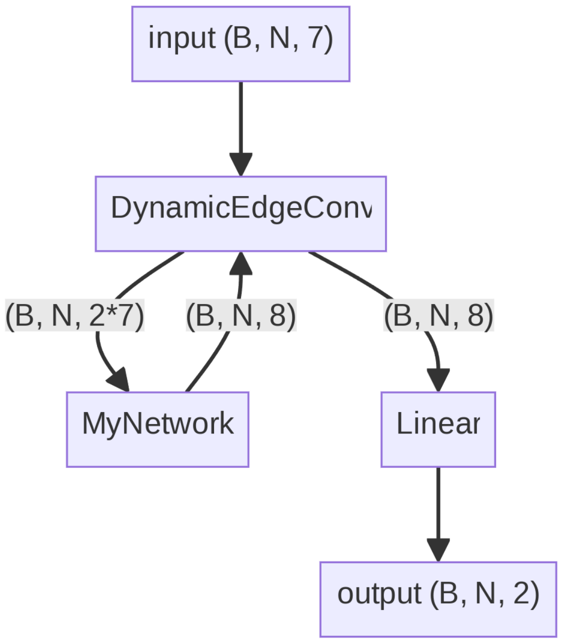

FIG. 6: Architecture of the Used Network

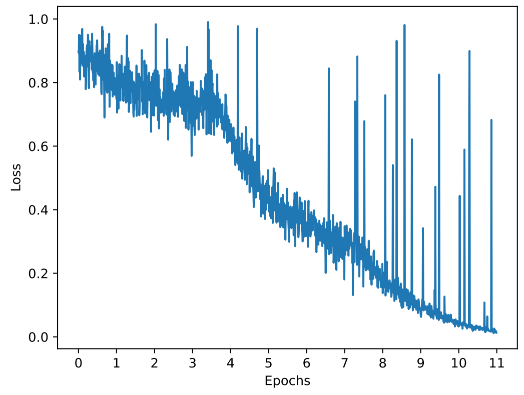

FIG. 7: Decreasing Loss with epoch

a final linear layer to produce the ultimate output of the model. The model architecture is given in Figure 6.

Here we should have predicted the output using clustering algorithms, but I didn't get enough time to do that. So, I am showing how my model is being able to differentiate between the foreground and the background clearly.

The model is trained using stochastic gradient descent (SGD) as the optimization algorithm, with a learning rate of 5×10^{-2}, momentum of 0.95, and weight decay of 0.0001. We used Binary-Crossentropy-Loss as the loss-function. Initially before applying smote, our loss was decreasing, but we were not getting very good results. This was because the data was not balanced, there were far more points in the background class than the foreground. Hence we applied resampling technique called SMOTE (explained in detail in II F). As a result, in figure 7 we see how the loss function decreases over training, and we also get good results to show (an example is shown is Figure 8)

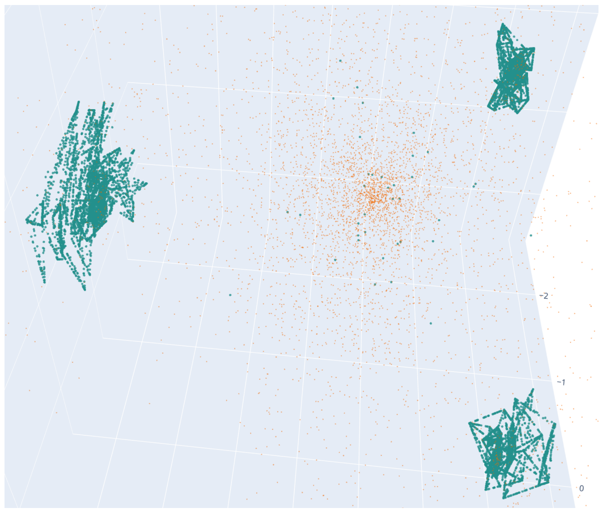

FIG. 8: Example Output From our model (green points have been labelled as foreground and orange points have been labelled as background by model)

IV. FUTURE WORK

While this study has made significant progress in developing a Graph Neural Network-based approach for point cloud classification, there are several avenues for future exploration.

1. Completing the instance segmentation task by extending the current framework to identify individual objects within the point cloud is a natural next step.

2. Running the same code on a larger dataset will help to generalize the results and increase the model's robustness.

3. Calculating proper metrics such as accuracy after prediction will enable a more comprehensive evaluation of the model's performance.

V. CONCLUSION

In conclusion, this project has been a valuable learning experience, providing insights into various aspects of point cloud processing and Graph Neural Networks (GNNs). Through this project, I had the opportunity to explore and learn about several new concepts and techniques.

1. Optimized searching algorithms like Octree and KDTree, which enabled efficient querying and processing of large point cloud datasets.

2. This project marked my first foray into GNNs, and I gained hands-on experience in designing and implementing a GNN-based approach for point cloud classification.

3. Working with huge point cloud data presented several challenges, and I learned effective strategies for handling and processing such datasets.

4. Additionally, I explored the concept of Siamese networks and their applications in computer vision tasks.

I would like to express my sincere gratitude to Prof. Subhankar Mishra and Prof. Subhashish Basak, for giving me the opportunity to work on this project, Mr. Jyotirmaya Shivottam for his immence guidance and Mr. Rhishav Das and Ms. Ipsita Rout for their support throughout this project. Their valuable insights and feedback were instrumental in shaping the project's outcome.

[1] V. Varatharasan, H.-S. Shin, A. Tsourdos, and N. Colosimo, Improving learning effectiveness for object detection and classification in cluttered backgrounds (2019) pp. 78–85.

[2] R. Castro, T. Lewiner, H. Lopes, G. Tavares, and A. Bordignon, Statistical optimization of octree searches, Comput. Graph. Forum **27**, 1557 (2008).

[3] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, Dynamic graph CNN for learning on point clouds, CoRR **abs/1801.07829** (2018), 1801.07829.

[4] G. Koch, R. Zemel, and R. Salakhutdinov, Siamese neural networks for one-shot image recognition (2015).

[5] A. Sharma, P. Singh, and R. Chandra, Smotified-gan for class imbalanced pattern classification problems, IEEE Access **10**, 1 (2022).

[6] F. Hu and H. Li, A novel boundary oversampling algorithm based on neighborhood rough set model: Nrsboundary-smote, Mathematical Problems in Engineering **2013** (2013).

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, Smote: Synthetic minority over-sampling technique, Journal of Artificial Intelligence Research **16**, 321–357 (2002).