Object detection in outdoor scene using VoteNet learning method

By:
Weisberg Ran
Final Project Report

GitHub Repository:
github.com/RanWeisberg/VoteNet-Custom-Dataset-and-DataGen

Advisor:
Schneor Ronit

Laboratory for CAD & Lifecycle Engineering
Faculty of Mechanical Engineering
Technion – Israel Institute of Technology
Haifa 32000, Israel

Abstract

This project explores the adaptation and evaluation of machine learning methods for the task of 3D object detection in outdoor environments. Specifically, it examines the VoteNet CNN learning model diverging it from its original indoor-focused application. The core objectives were twofold: generating a scalable synthetic dataset that simulates outdoor scenes to be used to train VoteNet. The dataset generation process included random objects placement in realistic LiDAR scans, and ground truth vote (tagging) according to the placement. These steps ensured realistic scene representation while adapting the dataset for effective model training.

However, the project encountered significant challenges, particularly in adapting the VoteNet model and generating the dataset. The dataset generation process proved to be a more complex and larger task than originally anticipated, requiring creative solutions to simulate realistic LiDAR scans and manage object placement within scenes. Additionally, adapting the model itself for use with outdoor datasets involved extensive modifications, including adjustments to training parameters and the loss function, which were not covered in the original Vote repository documentation.

Despite these efforts, the model's performance on outdoor scenes highlighted several limitations, including challenges in generalization, sensitivity to dataset characteristics, and suboptimal results in partial scans compared to full scene scans used in the original VoteNet paper. Unlike the original full scans, which captured entire rooms from multiple angles to provide a complete view of the scene and objects, the partial scans in this project simulated LiDAR data from a single point of view, offering only a projection of objects from the camera's perspective. These findings underscore the need for further research and optimization to enable robust outdoor object detection using VoteNet.

Contents

1 Introduction

The application of machine learning techniques in mechanical engineering has transformed the way complex problems are addressed, particularly in the realm of object pose estimation within three-dimensional (3D) environments. This project focuses on developing a process to adapt existing machine learning models for pose estimation in outdoor settings. The primary emphasis is on creating a scalable synthetic dataset that simulates outdoor scenes and establishing a robust methodology to train and test existing models using this dataset.

Outdoor environments pose significant challenges compared to controlled indoor conditions, such as laboratories or factories. While existing pose estimation models perform well under consistent conditions, their accuracy diminishes in outdoor settings due to dynamic and unpredictable factors like changing lighting, weather, and the presence of diverse and unexpected obstacles. Addressing these challenges requires not just robust models but also datasets that effectively mimic real-world outdoor scenarios.

To tackle these issues, this project adopts a two-pronged approach:

1. **Synthetic Dataset Creation:** Developing a scalable process for generating synthetic datasets that accurately simulate outdoor 3D scenes. This involves the placement of objects in realistic environmental settings.

2. **Model Adaptation:** Designing a pipeline to enable existing machine learning models to train on these datasets and evaluate their performance. This includes adapting the models to handle the unique characteristics of synthetic outdoor data and ensuring the testing framework is aligned with real-world application needs.

By focusing on these aspects, the project aims to bridge the gap between the capabilities of existing pose estimation models and the demands of outdoor environments. The creation of high-quality synthetic datasets not only facilitates efficient training but also allows for extensive testing and benchmarking without the need for labor-intensive data collection in the field.

This work contributes to improving object pose estimation in outdoor conditions, with potential relevance to fields like robotics and autonomous systems. The proposed process lays the groundwork for adapting existing models to dynamic real-world scenarios.

2 Literature Review

In this chapter I will cover existing methods that are relevant in the pipeline of the system such as, Capturing methods, 3D data representations, and neural networks.

2.1 Capturing Methods

Camera technologies for 3D image capture employ a variety of methodologies, from stereoscopic approaches to time-of-flight sensors, and structured light to light field cameras. Each technique offers a unique method for achieving depth perception.

- **Time Of Flight (ToF)**
  Time-of-Flight (ToF) technology determines depth by measuring the time it takes for light to travel from the camera to the subject and back. ToF cameras emit a light signal, often infrared, which reflects off objects in the scene and returns to the sensor. By calculating the travel time based on the known speed of light, ToF cameras provide accurate distance measurements for each point in the image, enabling the creation of detailed 3D representations [1].

  This technology is effective for real-time 3D imaging and is widely utilized in applications such as augmented reality, robotics, and gesture recognition [2]. However, ToF cameras can face challenges under certain environmental conditions, such as foggy or rainy environments, where the properties of light transmission are altered, affecting measurement accuracy [2]. Despite these limitations, ToF cameras remain a robust choice for applications requiring precise depth information.

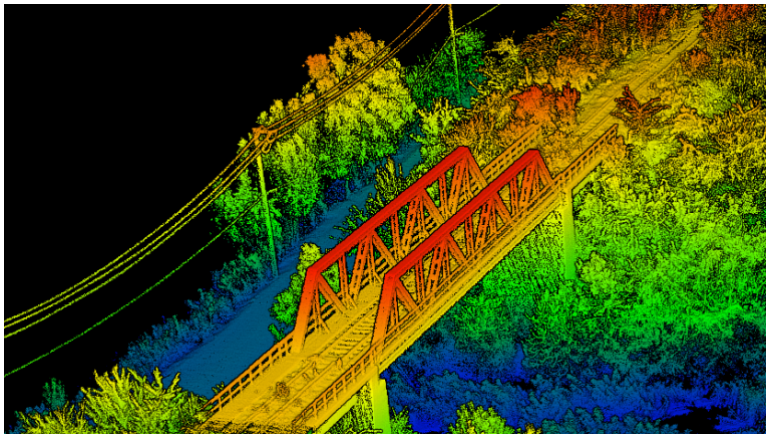


Figure 2.1: An example of a LiDAR Picture.

- **Structured Light**
  Structured light technology involves projecting a specific pattern of light, such as lines or grids, onto a scene and capturing the distorted pattern with a camera from a different angle. The deformation of the pattern, caused by the varying distances and shapes of objects, is analyzed to calculate depth and surface contours, resulting in a detailed 3D representation [3].

  This method is highly precise and suitable for applications requiring detailed surface reconstructions, such as quality control in manufacturing, facial recognition, and augmented reality. However, structured light systems typically require controlled lighting conditions to ensure the projected pattern is clearly visible, and they may struggle with highly reflective or transparent surfaces due to pattern distortion or loss [4].

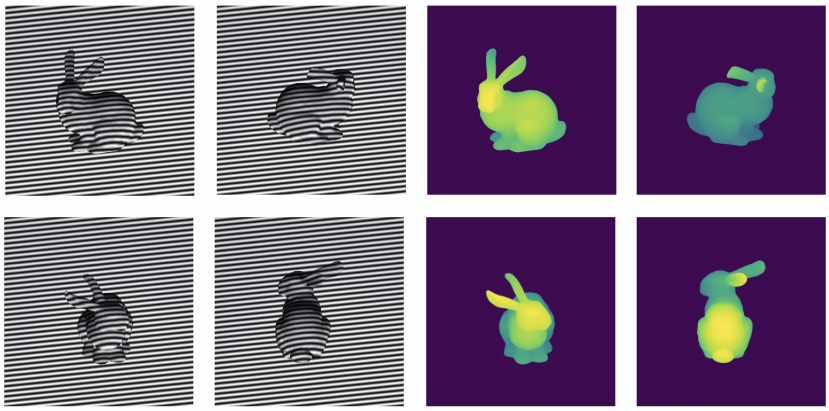


Figure 2.2: An example of Structured Light.

- **Stereoscopic Vision**
  Stereoscopic vision mimics the human visual system's depth perception by using two cameras positioned at slightly different angles to capture two separate images of the same scene. By comparing the differences between these images, the system calculates the depth of objects, creating a 3D representation [5].

  This method leverages the parallax effect—the apparent displacement of an object viewed along two different lines of sight—to gauge depth information. It is particularly useful in applications such as virtual reality, 3D filmmaking, and autonomous vehicle navigation. However, stereoscopic systems may face challenges in scenes with repetitive patterns or objects lacking distinctive features, which can make it difficult to match points between the two views [6].

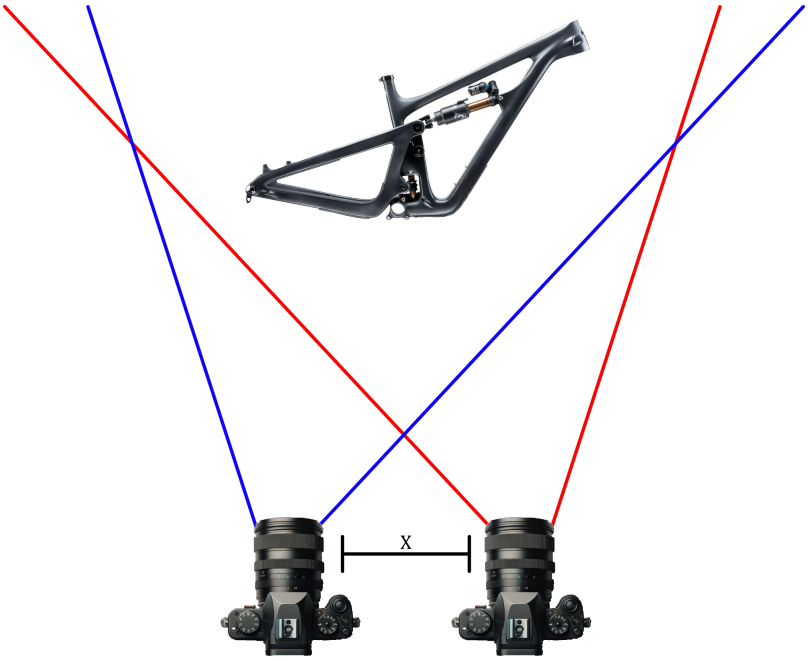


Figure 2.3: Stereoscopic Capturing Method.

2.2 3D Data Representations

In the realm of 3D data processing, transforming the output from capture technologies into practical models is critical. Various representation techniques offer different insights into the

captured data:

- **RGB-D Data**
  Merging color with depth data, RGB-D models capture the scene in vivid detail, presenting both the textural and spatial dimensions. These models are highly advantageous in interactive environments, providing the necessary depth cues for augmented reality and facilitating object recognition tasks in robotics. However, the technique may encounter difficulties with materials that do not reflect light consistently, such as transparent or shiny surfaces, and is typically constrained by the sensing range of the equipment [7].

- **Point Clouds**
  Constituting a collection of vertices in three dimensions, point clouds describe the external surface of objects with high precision. Essential for creating detailed environmental maps, they are the foundation for applications ranging from heritage conservation to autonomous navigation. While they excel in spatial resolution, point clouds often require substantial computational resources for processing and can be less effective in representing surface textures compared to other methods [8].

- **Voxel Models**
  Extending the 2D pixel concept into 3D space, voxel-based models offer a unique way to capture volumetric data, proving invaluable in medical imaging and architectural simulations. They allow for an all-encompassing representation, detailing not just the surface but also the internal structure of objects. The trade-off, however, comes in the form of high demand for storage and processing power, as the detailed volumetric data can quickly escalate in complexity [9].

- **Mesh Models**
  Built from vertices, edges, and faces to form a network, or 'mesh,' these models are staples in computer graphics, known for their efficiency in rendering complex shapes and surfaces. They are particularly favored in the entertainment industry for visual effects and animation. Mesh models can be less suited for scenarios requiring internal data representation and may also present challenges in accurately capturing intricate details without a significant increase in the mesh complexity [10].

2.3 Object Detection Using CNNs

Neural networks are sophisticated machine learning frameworks inspired by biological brain functions. These networks consist of interconnected layers of nodes, or "neurons," each capable of performing complex computations. As data passes through these neurons, it is processed based on learned weights and biases, enabling the network to detect patterns and make informed predictions. This adaptability makes neural networks particularly effective in diverse applications, from image recognition to autonomous systems. The following sections will explore specific neural network configurations such as VoteNet, PointNet++, YOLO 6D, and EfficientDet, which leverage this technology to recognize objects in 3D space, demonstrating their utility in integrating with 3D cameras for advanced spatial analysis.

- **VoteNet** is a deep learning model specifically designed for object detection in 3D point clouds. [11] Developed by Facebook AI Research, it builds on the PointNet++ backbone to process sparse 3D points effectively. VoteNet employs a unique voting mechanism, where each seed point in the point cloud casts "votes" for the center of the object it belongs to. These votes are generated by predicting offsets from the seed point to the object's center, known as **ground votes**, which are derived from the ground truth bounding boxes during training.

The architecture of VoteNet is particularly well-suited for point cloud data obtained from 3D sensors such as LiDAR and depth cameras. It excels at detecting objects in unstructured 3D environments by leveraging the following key concepts:

- **Seed Points:** A subset of points sampled from the input point cloud that forms the basis for generating votes.
- **Voting Module:** A neural network module that predicts offset vectors from seed points to the object centers. The predicted votes cluster around these centers, enabling the detection module to locate objects effectively.
- **Bounding Box Proposal:** From the clustered votes, the network generates 3D bounding box proposals that describe the objects' location, size, and orientation.
- **Ground Votes:** During training, these are the offset vectors calculated from seed points to the actual object centers (ground truth). The loss function minimizes the difference between predicted votes and ground votes, ensuring accurate center predictions.

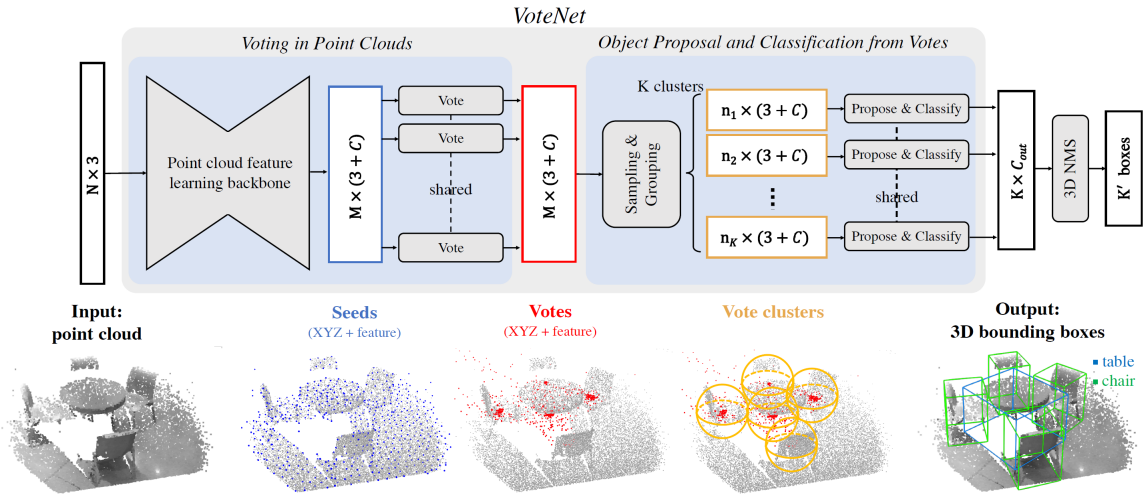


Figure 2.4: Illustration of the VoteNet architecture from VoteNet paper.

- **PointNet++** is an advanced version of the original PointNet, enhancing its capability to capture local structures in point clouds by applying a hierarchical neural network that processes points in nested partitions [12]. This makes PointNet++ adept at handling non-uniformly sampled data, commonly produced by 3D cameras, and is typically used in semantic segmentation and object classification tasks.
- **YOLO 6D** extends the original YOLO (You Only Look Once) framework to the realm of 6D object pose estimation [13]. This variant predicts both the location and orientation of objects in 3D space using a single network pass, making it highly efficient for real-time applications. It is optimized for RGB-D images, where it leverages depth data for better spatial understanding.
- **EfficientDet** is a scalable and efficient object detection model that, while primarily developed for 2D images, has been adapted for 3D object detection when combined with depth-aware convolution layers. This adaptation allows it to process 3D data efficiently, providing a balance between speed and accuracy for detecting objects in volumetric spaces.

2.4 3D Object Datasets

In the field of 3D object detection and scene understanding, several datasets have become benchmarks, providing diverse and challenging environments for testing and training models. Among these, the KITTI and SUN-RGBD datasets stand out for their comprehensive coverage of various real-world scenarios. This subsection explores these datasets and others, highlighting their unique contributions to the field.

- **KITTI Dataset**
  The KITTI Vision Benchmark Suite, introduced by Geiger et al. [14], is a widely used dataset for 3D object detection and tracking, visual odometry, and scene flow estimation. The dataset comprises high-resolution images and 3D point clouds captured using a variety of sensors, including stereo cameras and LiDAR. KITTI covers urban, rural, and highway scenarios, providing a comprehensive range of scenes.

  The dataset's point cloud data, generated from LiDAR scans, offers accurate 3D representations of the captured scenes. KITTI's annotations include 3D bounding boxes, which facilitate the training and evaluation of 3D object detection algorithms. Despite its extensive coverage, KITTI's primary limitation lies in its outdoor focus, with less emphasis on indoor environments. Nevertheless, it remains a critical resource for autonomous driving research, providing robust benchmarks for various tasks.

- **SUN-RGBD Dataset**
  The SUN-RGBD dataset, developed by Song et al. [15], provides a comprehensive collection of indoor scenes, captured using multiple sensors, including Kinect, Intel RealSense, and Asus Xtion. The dataset includes RGB-D images, with detailed annotations for over 10,000 object categories, making it one of the most extensive datasets for indoor scene understanding.

  SUN-RGBD's annotations encompass 3D bounding boxes and 2D segmentation masks, enabling a wide range of tasks, including object detection, segmentation, and scene classification. The diversity of sensor types and the variety of indoor environments captured—ranging from living rooms to offices—make SUN-RGBD an essential dataset for indoor navigation and robotics. However, the dataset's complexity can pose challenges in terms of annotation consistency and depth accuracy, especially when using different sensor modalities.

- **NYU Depth Dataset V2**
  The NYU Depth Dataset V2, introduced by Silberman et al. [16], is another prominent dataset focused on indoor scenes. Captured using a Microsoft Kinect, this dataset includes RGB-D images from various indoor environments, along with dense 3D annotations for over 1,000 objects. The dataset also provides pixel-level segmentation for 40 object categories, facilitating fine-grained scene understanding.

  The NYU Depth Dataset V2's detailed annotations and rich depth information make it a valuable resource for developing algorithms for object detection, scene segmentation, and depth estimation. Its focus on indoor environments complements the outdoor emphasis of datasets like KITTI, providing a balanced perspective for 3D scene understanding. However, like many datasets, it faces challenges related to sensor noise and the limited field of view of the Kinect.

- **ScanNet**
  ScanNet, introduced by Dai et al. [17], offers an extensive dataset of indoor scenes captured using a low-cost RGB-D sensor. The dataset includes over 1,500 scans, each with complete surface reconstructions and object-level annotations. ScanNet provides not only RGB-D

frames but also 3D surface reconstructions, enabling detailed 3D modeling and semantic understanding.

ScanNet's comprehensive annotations include 3D bounding boxes, surface meshes, and semantic labels, making it a versatile dataset for tasks such as 3D object detection, semantic segmentation, and 3D reconstruction. Its focus on dense scene capture and detailed object-level annotations provides a rich resource for research in indoor scene understanding. However, the dataset's reliance on low-cost sensors can introduce noise and inaccuracies in depth data, presenting challenges for precise 3D modeling.

2.5 Loss Functions in Point Cloud Models

Loss functions are essential for training models to accurately interpret and represent 3D data. They quantify the difference between predicted outputs and ground truth, guiding the optimization of model parameters. Key loss functions in point cloud models include:

- **Vote Loss**
  Vote loss measures the discrepancy between predicted vote points and their corresponding object centers. In architectures like VoteNet, seed points generate votes that predict object centers. The vote loss penalizes deviations between these predicted votes and the actual object centers, ensuring that votes accurately converge towards the correct locations [11].

- **Objectness Loss**
  Objectness loss evaluates the probability that a point belongs to an object versus the background. It is typically implemented using binary cross-entropy loss, distinguishing points that are part of objects from those that are not. This loss guides the model to focus on regions of interest within the point cloud [11].

  Objectness Class Weights
  To address the imbalance between object and background points, objectness loss often incorporates class weights. These weights amplify the contribution of less frequent object points, ensuring that the model does not disproportionately focus on the majority background class. By scaling the loss for object points relative to background points, the model learns to identify objects more effectively, even in datasets with a significant background-to-object ratio [11].

- **Box Loss**
  Box loss pertains to the accuracy of predicted bounding boxes around objects. It encompasses errors in parameters such as center coordinates, dimensions, and orientation. Loss functions like smooth L1 or L2 are commonly used to minimize these errors, ensuring precise localization of objects within the point cloud [11].

- **Semantic Classification Loss**
  This loss function assesses the model's ability to classify points or regions into semantic categories (e.g., chair, table). Typically implemented using cross-entropy loss, it penalizes incorrect classifications, enabling the model to differentiate between various object classes within the point cloud [11].

In training point cloud models, maintaining a balance between positive and negative samples is crucial:

- **Positive (Pos) Ratio and Negative (Neg) Ratio**
  The positive ratio refers to the proportion of samples that correspond to actual objects, while the negative ratio pertains to background or non-object samples. An imbalance, such as an excess of negative samples, can bias the model, leading to poor detection performance.

Strategies like hard negative mining or focal loss are employed to address this issue, ensuring the model learns effectively from both positive and negative samples [11].

2.6 Performance Metrics for Object Detection

Evaluating the performance of object detection models requires robust metrics that capture their accuracy and reliability. These metrics are often derived from the confusion matrix, a summary table that compares predicted outputs with ground truth. The key parameters of the confusion matrix include:

- **True Positives (TP):** Cases where the model correctly identifies an object that is present in the data [18].
- **False Positives (FP):** Cases where the model predicts an object that is not present (also referred to as a "false alarm") [18].
- **True Negatives (TN):** Cases where the model correctly identifies the absence of an object [18].
- **False Negatives (FN):** Cases where the model fails to detect an object that is present in the data [18].

Using these parameters, two critical metrics—**precision** and **recall**—are calculated to assess the model's performance in terms of its correctness and sensitivity:

- **Precision:** This metric measures the proportion of correctly predicted positive cases (true positives) out of all positive predictions made by the model. It is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

  Precision indicates how often the model's predictions are correct when it predicts the presence of an object. A high precision score signifies that the model produces fewer false positives, which is critical in applications where false alarms carry high costs [19].

- **Recall:** This metric, also known as sensitivity, measures the proportion of actual positive cases (true objects) correctly identified by the model. It is defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

  Recall assesses the model's ability to detect objects. A high recall score indicates that the model successfully identifies most of the objects present in the data, minimizing false negatives [19].

2.7 Z-Depth Buffering

Z-depth buffering is a core technique used in computer graphics to manage depth information and ensure correct rendering of objects in a 3D scene. This process is fundamental for applications like occlusion handling and realistic perspective projection. The Z-buffer, a dedicated data structure, stores depth values corresponding to each pixel on the screen, which are derived from the camera's viewpoint.

When rendering a scene, each object is projected into 2D screen space, and its depth information is calculated for each pixel it occupies. The Z-buffer then checks whether the newly calculated depth for a pixel is closer to the camera than the existing value stored in the buffer.

If the new depth is closer, the Z-buffer updates the value, and the pixel's color is rendered. Otherwise, the new depth is discarded, as it is occluded by another object.

This concept is illustrated in Figure 2.5, where the top image shows a 3D scene containing a monkey head, a green sphere, and a cube. The bottom image visualizes the Z-buffer's depth representation, where darker shades indicate greater depth (objects farther from the camera). This mechanism ensures that only the visible surfaces of objects are rendered, and occluded parts are hidden from view.

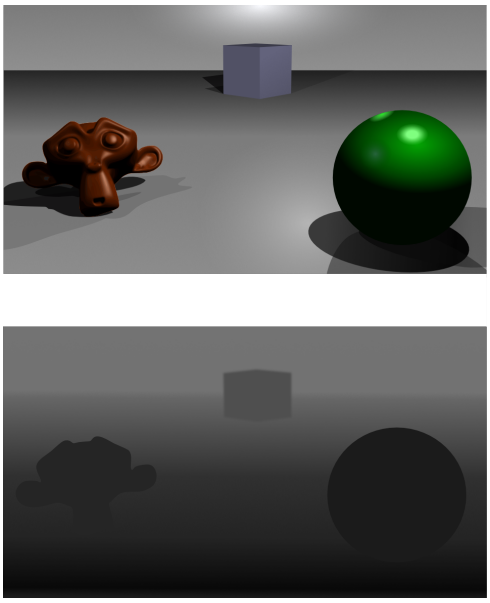


Figure 2.5: Illustration of Z-buffering. The top image displays a 3D scene, while the bottom image shows the corresponding Z-buffer depth map.

Z-depth buffering is essential for maintaining geometric consistency and realism in rendered scenes, and its principles have broad applicability in both computer graphics and data processing workflows [20].

3 Problem Statement

In the autonomous systems and machine learning, the identification and orientation (pose estimation) of objects within three-dimensional environments pose significant challenges, especially in dynamic and uncontrolled outdoor settings. These challenges are compounded by variable factors such as lighting and weather conditions, which can severely impact the performance of traditional detection systems optimized for static, controlled environments. Current technologies, while proficient within predictable confines, often fail to maintain accuracy and reliability when subjected to the complexities of outdoor scenarios.

The objective of this project is to adapt and evaluate the VoteNet model for 3D object detection in outdoor environments. This involved generating a synthetic dataset that simulates outdoor scenes using partial LiDAR scans and modifying the VoteNet pipeline to process this dataset effectively. The project aimed to assess the model's performance in detecting and localizing objects under challenging outdoor conditions and to identify limitations in its generalization and robustness, paving the way for future improvements in outdoor object detection systems.

This problem statement encapsulates the core challenges and technological gaps that this project seeks to address, emphasizing the impact of environmental variability on the effectiveness of pose estimation techniques and the need for advanced solutions in practical, real-world applications.

4 Approach

The primary objective of this project is detecting objects in an outdoor environment scene captured by a LiDAR camera and using the VoteNet learning model. Specifically, the project aims to assess the performance of VoteNet in accurately identifying objects and estimating their poses in simulated real-world outdoor scenarios. This evaluation involves creating a comprehensive synthetic dataset that mirrors real-world outdoor conditions to be used for training. In this dataset, various objects are randomly positioned and oriented within the scene to reflect the unpredictable nature of real-world environments.

To enhance realism, the scenes undergo modifications to simulate conditions such as projections, where the perspective may alter the appearance of objects. Additionally, the process of data generation and model adaptation has been refined to increase flexibility. Unlike the original setup, which was limited to specific datasets with predefined scripts, the system supports training and prediction with any 3D object represented in STL format, as long as it fits within the scenes. This enhancement is achieved by developing a more modular data generation pipeline and modifying the model to seamlessly accept the new data format.

The data generation process, which forms the backbone of this project, is visually summarized in Figure 4.1. The diagram outlines the key steps in the pipeline, starting with the input of 360° LiDAR point cloud scans from the KITTI dataset. These scans are divided into smaller segments, and scene boundaries are defined to prepare them for object placement. The `place.py` module is then invoked, which handles object model loading, random placement, and feasibility checks to avoid overlaps. Simulated LiDAR scans are generated for the placed objects, which are then integrated into the final scene. The process concludes with dataset configuration and export, ensuring compatibility with VoteNet.

Through this approach, we aim to provide a thorough evaluation of VoteNet's effectiveness in complex outdoor scenarios, highlighting its strengths and identifying potential areas for improvement. The insights gained from this study will contribute to the broader field of computer vision and autonomous systems, particularly in applications involving outdoor object recognition and interaction.

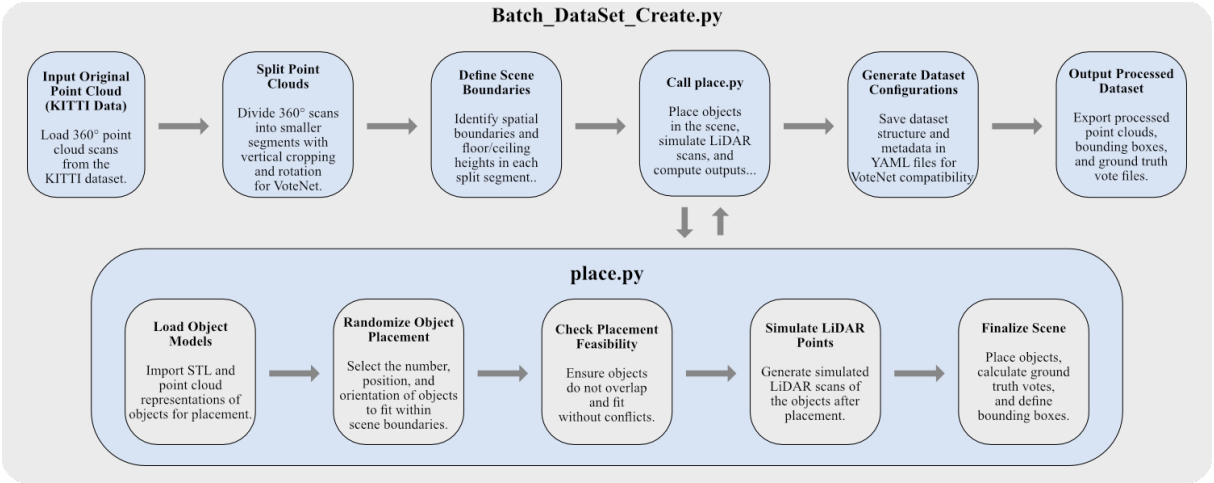


Figure 4.1: Overview of the data generation process.

5 Implementation

As explained in the approach, the proposed pipeline includes two main tasks: (1) generating an outdoor training dataset and (2) implementing the VoteNet model.

5.1 Data Generation Process

To create a synthetic dataset suitable for use with VoteNet, and ensure it is large enough to effectively train the model on a diverse range of objects in simulated outdoor real-world scenarios, the data generation process was divided into several key steps. These steps were designed to replicate the complexity and variability of real-world environments, providing a comprehensive foundation for model training and evaluation. The following sections detail each stage of this process, from object selection and placement to environmental condition simulation and camera emulation.

5.1.1 KITTI Dataset Adaptation

To create a sufficiently large dataset, we utilized the raw data from the LiDAR scans provided by the KITTI Vision Benchmark Suite [14]. As shown in Figure 5.1, these scans cover a full 360-degree view. To adapt this data to simulate the output of an Intel RealSense D435 camera, we performed several key steps.

First, to match the camera's field of view (FOV), the 360-degree scans were split into five distinct sections, each with a horizontal FOV of 69 degrees and a vertical FOV of 42 degrees. This splitting process not only increased the dataset size but also generated files that closely resemble the real-world scans expected from the target camera. An added advantage of using the KITTI scans is that they already represent real-world conditions, thus bypassing the need for additional steps to simulate realistic environments.

Another crucial step was transforming the coordinate system from the one used by the KITTI dataset to the format required by the VoteNet model. Specifically, the transformation involved mapping the original y-axis to the x-axis, the original x-axis to the y-axis, while the original z-axis remained unchanged. This transformation resulted in five new scenes that accurately simulate real-world scans taken by the Intel RealSense D435 camera, presented in the coordinate system needed for VoteNet.

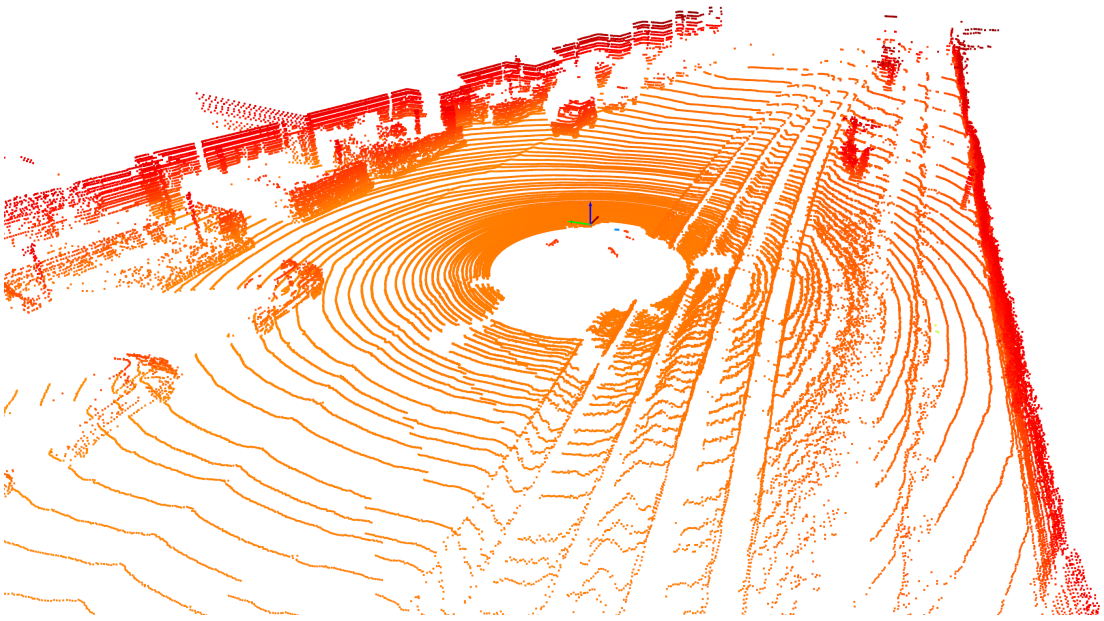


Figure 5.1: A Sample Image of a Scene from KITTI Raw Dataset.

5.1.2 Object Selection and Preprocessing

To generate a new labeled dataset featuring random placements of objects in space, it was crucial to select objects that would be appropriately scaled for the scenes. Additionally, the selected objects needed to be different from those found in the original dataset to prevent the model from mistakenly recognizing objects not included in the labels.

In this project, five public STL models were chosen due to their low likelihood of appearing in the original scenes. These models required further preprocessing, as they were not originally created for this specific task.

To ensure consistency across the models, each was adjusted so that its origin was moved to the base of the model. This was accomplished by developing a script that shifted the origin based on the desired values, which were measured using SolidWorks.

Moreover, the models were scaled to sizes that would match the proportions seen in KITTI scans. After scaling, another script converted the models from STL format to point clouds and adjusted the directions of the coordinate system to align with the coordinate system used in this project.

The table below presents the selected models and their corresponding scaling factors.

Model	**RubberDuck**	**Shuttle**	**F1Car**	**CoffeeMug**	**Benchy**
					
Scaling Factor	15	17	30	30	30

Table 1: Selected models and their scaling factors.

5.1.3 Object Placement and Scene Boundary Definition

To create the dataset, a script was developed to randomly place between 1 to 6 of the modeled objects within each scene, assigning them random positions and rotation around the z-axis. This randomness simulates a variety of real-world conditions. Several functions were implemented to ensure that the objects were placed correctly within the scene boundaries.

The process began with defining the scene's horizontal and vertical boundaries. A function was used to calculate the scene's horizontal boundaries by identifying the minimum and maximum x and y values in the scene, forming a rectangular bounding area. To determine the vertical boundaries, another function calculated the floor height using a histogram of the z-values, where the peak represented the floor. The ceiling was defined by the maximum z-value within the scene.

Once the scene boundaries were defined, objects were positioned within the scene using random translations and rotations. For each object, a random heading angle was generated, and a corresponding rotation matrix was applied. The object was then translated to a random position within the scene's boundaries, ensuring that the object fit within the defined height limits.

Before finalizing the placement, the script checked for potential overlaps between the object's transformed point cloud and the existing points in the scene. If a collision was detected, the script would attempt to place the object in a different location and orientation, repeating the process until a non-overlapping placement was found or the maximum number of attempts was reached.

Once successfully placed, a partial LiDAR scan of the object was simulated by generating a depth buffer from the object's STL file, which was then transformed and added to the scene. The object's bounding box parameters were calculated, including its center, size, and combined heading angle. The bounding box data, along with the object class label, was stored for later use in model training.

Additionally, the script adjusted the size of the combined point cloud to a target number of points, ensuring consistency across different scenes. Ground truth votes were computed for each point in the scene, indicating the direction and distance to the center of the nearest object bounding box. This information was saved, along with the final scene and label data, for training the VoteNet model.

5.1.4 Simulated LiDAR Scan Generation

To simulate the LiDAR scans, a specialized script was developed to generate realistic point cloud data from the modeled objects. This process involved calculating the appropriate angle at which the object should be scanned based on its placement relative to the origin of the scene, effectively mimicking a real-world LiDAR scanner's behavior.

The previous script first determined the object's position and orientation within the scene, including its rotation around the z-axis. From this information, the script calculated the angle at which the object should be scanned. This combined angle was derived from the object's placement relative to the scene's origin and its rotation, ensuring that the simulation accurately reflected the scanning process as if a LiDAR scanner were capturing the object from the scene's center.

Once the scanning angle was determined, another script took over. Using OpenGL, this script placed the object at the origin of the virtual environment and positioned a virtual camera at a fixed distance. The camera was then rotated around the object based on the previously calculated angle, simulating the movement of a real LiDAR scanner. The script utilized z-depth buffering to capture depth information, ensuring that only the points visible from the specific angle were recorded.

This depth data was then saved in point cloud form, representing the simulated scan from that particular angle. The resulting point cloud, which contained only the visible points from

the simulated LiDAR perspective, was returned to the previous script. This point cloud was then integrated into the scene, completing the simulation of a LiDAR scan as if the objects had been scanned in a real-world environment.

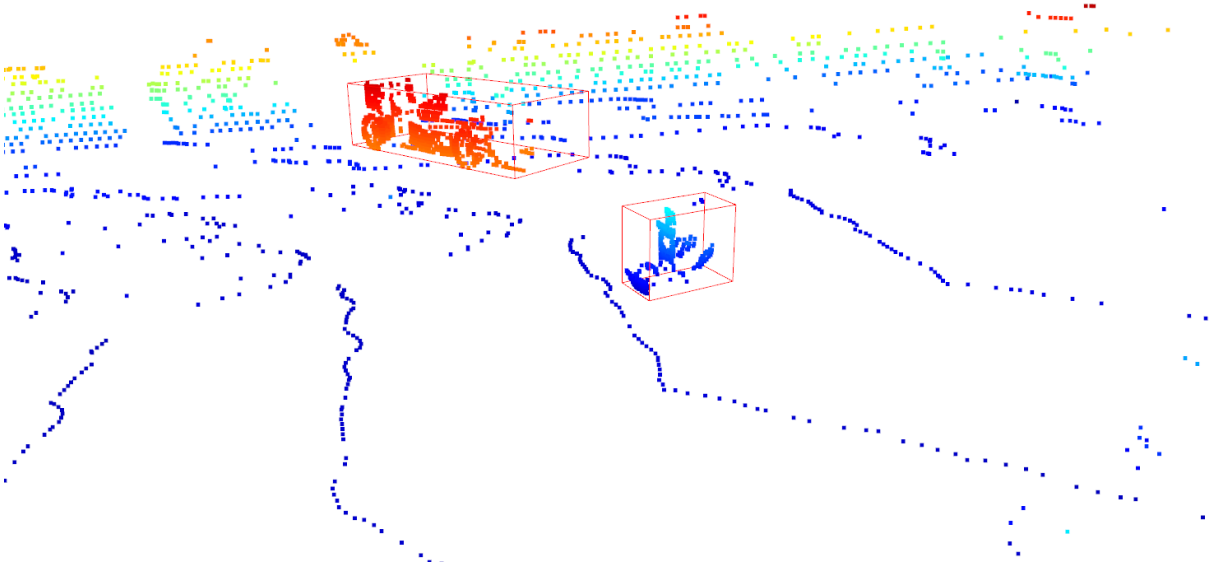


Figure 5.2: Example of simulated object placement within the scene with corresponding bounding boxes.

The figure above illustrates an example of how the objects were placed within the scene and their corresponding bounding boxes. The color variation in the point cloud data represents different depth levels, showcasing the accuracy of the simulated LiDAR scan in capturing the scene.

5.1.5 Dataset Generation and Organization

Finally, a main script was developed to automate the entire process across the entire KITTI dataset. This script systematically processes the data, creating a comprehensive dataset divided into 80 percent for training, 10 percent for validation, and 10 percent for testing. The script ensures that all necessary files are generated and organized according to the structure required to work with the VoteNet model.

After running the script, the resulting dataset sizes were as follows: 33,228 scenes for training, 4,154 for testing, and 4,154 for evaluation. For each scene in the subsets (training, validation, and testing), the script generated three key files:

- **Bounding Boxes:** Defined using the format (x, y, z) for the center, (l, w, h) for the size, and the heading angle of the object.
- **Ground Truth Votes:** Including offsets from seed points to object centers as well as information indicating whether a point is part of an object or the background.
- **Point Cloud Data:** Representing the 3D spatial structure of the scene.

In addition to these per-scene files, the script also generated a single comprehensive configuration file for the entire dataset. This configuration file contains relevant metadata and parameters required for seamless integration into the training pipeline, ensuring the model can utilize the dataset effectively.

5.2 Model Adaptation

To enable VoteNet to function effectively with a custom outdoor dataset, significant modifications were made to its structure and training process. These adaptations were necessary to address the unique characteristics of the dataset, including object types, sizes, and the nature of partial LiDAR scans. By integrating custom configurations, adjusting loss functions, and tailoring the training pipeline, the model was optimized for outdoor object detection in challenging environments.

5.2.1 Dataset Integration

To enable the VoteNet model to operate with a wider range of datasets and to provide greater flexibility, two new scripts were developed, inspired by the existing scripts used for the ScanNet dataset. These scripts facilitate seamless integration between the dataset generation process and the VoteNet model by ensuring compatibility with any 3D dataset represented in STL format.

- **CustomDatasetConfig:**
  - This script dynamically configures dataset parameters, ensuring flexibility and modularity. By reading a `data_config.yaml` file generated during the dataset creation process, the script allows the VoteNet model to adapt to different datasets without requiring hardcoded configurations.
  - Key attributes such as the number of classes, heading bins, size clusters, and object types are loaded from the configuration file. Essential mappings computed by the script include:
    - `type2class`: Maps object types to class indices.
    - `type_mean_size`: Stores the mean sizes of each object type for accurate bounding box calculations.
    - `size2class` and `class2size`: Enable conversions between object sizes and class-specific residuals.
    - `angle2class` and `class2angle`: Handle the discretization of object orientations into heading bins.
  - This modular approach ensures that any dataset conforming to the specified structure can be seamlessly integrated into the model.
- **CustomDetectionDataset:**
  - This script serves as the primary data loader for the VoteNet model. It processes the point cloud data, bounding box labels, and ground truth votes generated during dataset creation, preparing them for training and inference.
  - The script supports configurable attributes such as the number of points per scene, data augmentation, and height-based feature computation. For each scene, it:
    - Loads point cloud data, ensuring consistent point sampling.
    - Prepares bounding box labels, including object centers, sizes, orientations, and semantic classes.
    - Loads ground truth votes, which indicate the relationship between points and object centers.
    - Applies data augmentation techniques like flipping and rotation to enhance the model's generalization ability.

5.2.2 Loss Function Modifications

The loss function in the original VoteNet model was adapted to enhance its performance in detecting objects within the newly generated dataset. Several key changes were made to the loss calculation to align it with the characteristics of the dataset and improve its robustness in outdoor environments:

Threshold Adjustments: The thresholds for defining the proximity of predicted object centers to ground truth centers were modified:

- **Near Threshold:** Increased from 0.3 to 1.8.
- **Far Threshold:** Increased from 0.6 to 2.0.

These adjustments were made to reflect the specific sizes of the objects in the dataset, ensuring that the detection thresholds account for the larger spatial dimensions and object scales. This change reduces sensitivity to small positional discrepancies while maintaining accurate center predictions.

Objectness Class Weights: The objectness classification weights were recalibrated to reflect the distribution of object and background samples in the dataset:

- **Original Weights:** [0.2, 0.8] (Background, Object)
- **Modified Weights:** [0.1, 1.0] (Background, Object)

To determine these weights, a custom script was developed and utilized to compute the average ratio of object samples to background samples across the dataset. This script analyzed the `point_votes` data within the '.npz' files generated during dataset creation. For each file, it calculated the ratio of object points (mask value = 1) to the total number of points (object + background). The ratios were then averaged across all files, providing a dataset-wide object-to-background ratio.

This data-driven approach ensures that the objectness weights reflect the true prevalence of object and background points in the dataset. By giving more weight to object detection, the model is better equipped to handle imbalanced datasets, where object points may be relatively sparse compared to background points.

Loss Weights: The relative contribution of objectness loss in the overall loss function was increased:

- **Original:** $\text{loss} = \text{vote\_loss} + 0.5 \times \text{objectness\_loss} + \text{box\_loss} + 0.1 \times \text{sem\_cls\_loss}$
- **Modified:** $\text{loss} = \text{vote\_loss} + 2.0 \times \text{objectness\_loss} + \text{box\_loss} + 0.1 \times \text{sem\_cls\_loss}$

This adjustment emphasizes the importance of accurate objectness classification, especially in challenging outdoor scenarios with variable object density.

Confusion Matrix Calculation: A new addition to the loss calculation was the computation of a confusion matrix for objectness predictions, including:

- True Positives (TP)
- False Positives (FP)
- True Negatives (TN)
- False Negatives (FN)

These metrics provide detailed insights into the model's objectness classification performance, enabling a deeper understanding of its strengths and areas for improvement.

5.2.3 Training Script Modifications

The training script for the VoteNet model was significantly modified to accommodate the unique requirements of the custom dataset and to enhance its functionality and robustness during training. These modifications aimed to streamline the training process while providing additional insights into model performance. Key changes and improvements include:

Dataset Integration: The script was adapted to work with the custom dataset by replacing references to the original datasets (e.g., SUN RGB-D or ScanNet) with the custom dataset configuration and dataloader. This integration ensures seamless compatibility with the dataset generated in the data preparation pipeline.

Logging Enhancements: Comprehensive logging mechanisms were introduced to improve the visibility of training and evaluation processes:

- **CSV Logging:** Metrics for each training and evaluation epoch, such as loss components, objectness accuracy, positive and negative ratios, and confusion matrix values (e.g., TP, FP, TN, FN), are saved to CSV files. This provides a structured and persistent record of model performance.
- **Real-Time Visualization:** Integration with TensorBoard for real-time monitoring of training metrics, enabling immediate feedback on model behavior.

Performance Metrics: The script computes detailed metrics, including a confusion matrix (true positives, false positives, true negatives, and false negatives) and ratios of positive to negative predictions. These metrics provide deeper insights into the model's classification behavior, especially in imbalanced datasets.

Checkpointing and Resumption: Enhanced checkpointing mechanisms allow the training process to resume effectively:

- **Selective State Loading:** Checkpoints are validated for parameter compatibility with the current model. Mismatched parameters are reinitialized to ensure model consistency, while compatible parameters are restored.
- **Loss Tracking:** The best validation loss is tracked and used to save the best model checkpoint, ensuring that the most effective model is preserved.

Runtime Estimation: Additional functionality estimates the total runtime and provides real-time predictions of the remaining time required to complete the training process. This feature aids in resource planning and experiment management.

6 Model Training

The training process was designed to evaluate the performance of the adapted VoteNet model using the custom outdoor dataset. This section outlines the key training configurations, including dataset specifics, hyperparameter settings, and hardware requirements. It also addresses challenges such as class imbalance and the model's sensitivity to outdoor data, with tailored adjustments to improve training efficiency and outcome reliability.

6.1 Training Parameters

The training process was configured using the some defult parameters from the original model, and some accived after some trail and error. These parameters controlled the dataset, model, and optimization strategies, ensuring the training procedure aligned with the requirements of the project. Below is a summary of the key configurations:

6.1.1 Key Parameters

- **Dataset:** Custom dataset.
- **Model:** `VoteNet`, with modifications for compatibility with the dataset and training requirements.
- **Batch Size:** 8
- **Number of Points:** 20,000 per point cloud.
- **Number of Proposals:** 256
- **Learning Rate:** Initial learning rate set to 0.001, with decay applied at epochs 70, 100, and 140.
- **Max Epochs:** 150
- **Optimizer:** Adam, with no weight decay.
- **Batch Normalization Decay:** Reduced from 0.5 to 0.001 over 20-epoch steps.

6.1.2 Hardware Setup

The training process was conducted on the following hardware configuration:

Component	**Details**
GPU	Nvidia RTX 2060 (6GB VRAM)
Additional Memory	15GB shared system memory
GPU Usage	6-7GB during training

Table 2: Hardware setup for training.

6.1.3 Training Configuration Details

The script was executed with various features, including:

- **Height Signal:** Enabled unless explicitly disabled via `-no_height`.
- **Cluster Sampling:** Configured as `vote_fps` for optimal proposal generation.
- **Evaluation:** Performed after each epoch, computing metrics such as precision, recall, and confusion matrices.

6.2 Addressing Bias in Classification

During the initial phase of model training, the loss function was left unmodified as it was assumed that the original implementation would generalize effectively to the custom dataset. Various training parameters were tuned to optimize performance, including the learning rate, batch size, and decay rates. Despite these adjustments, the model consistently converged to an outcome where nearly **100% of the positive-to-negative ratio (pos_ratio)** was negative, indicating that the model classified nearly all points as background.

This behavior revealed a significant issue: the model had developed a bias toward classifying all points as background, effectively neglecting the identification of objects.

6.2.1 Root Cause Analysis

To understand the cause of this behavior, an in-depth analysis of the loss calculation script was conducted. It was discovered that the mismatch between the **ratio of object points to background points in the dataset** and the calibrated weights in the loss function led to this issue. Specifically:

- The loss function was calibrated with **objectness class weights** that heavily favored background classification.
- Due to the dataset's inherent imbalance (a significantly higher number of background points compared to object points), the model learned that the penalty for misclassifying object points as background was minimal. This encouraged the model to classify all points as background to minimize loss.

6.2.2 Addressing the Issue

To resolve this bias, the **object-to-background point ratio in the dataset** was analyzed using the custom script mentioned earlier. The script computed the ratio of object points to background points for all scenes in the dataset. Based on these findings, the **objectness class weights** were recalibrated to reflect the actual dataset distribution, significantly increasing the weight for object points.

Additionally, the loss function was further modified to address other minor issues discovered during analysis, including adjusting the **FAR_THRESHOLD** and **NEAR_THRESHOLD** values to better align with the dataset's spatial characteristics.

6.2.3 Outcome

After implementing these changes to the loss function, the specific issue of classifying all points as background was resolved. The model began to properly balance object identification with background classification, resulting in meaningful predictions during training.

6.3 Training Results

The training process took 114.27 hours to complete. During this time, performance metrics were recorded at each epoch in the `train_log.csv` and `eval_log.csv` files. This subsection analyzes key metrics derived from the training process to assess the model's performance and convergence.

6.3.1 Training and Evaluation Loss Analysis

During the training process, the model's performance was monitored by tracking both training and evaluation loss metrics. Figure 6.1 and Figure 6.2 depict the respective loss trends over the course of 150 epochs.

Training Loss The training loss exhibited a consistent decline over the training epochs, as shown in Figure 6.1. Initially, the loss reduced rapidly, followed by a more gradual decrease, eventually plateauing. This trend aligns with the learning rate decay schedule applied during training. Each decay step corresponded to a slower reduction in loss, suggesting that the model gradually reached its capacity to minimize the loss under the current configuration. This observation implies that extending the training duration or using a slower decay rate could potentially yield further improvements.

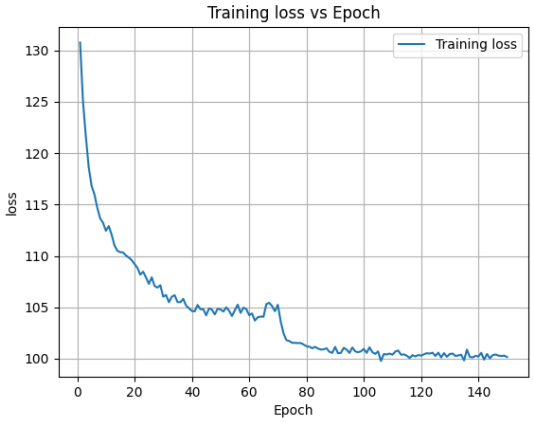


Figure 6.1: Training loss over 150 epochs.

Evaluation Loss The evaluation loss, presented in Figure 6.2, displayed significant fluctuations throughout the training process. Unlike the training loss, which decreased steadily, the evaluation loss lacked a clear downward trend, suggesting potential issues with the model's generalization ability. The oscillatory behavior of the evaluation loss may be attributed to overfitting, wherein the model becomes increasingly attuned to the training data at the expense of performance on unseen data.

Additionally, it is important to note that the scaling factors for the total loss differ between training and evaluation, resulting in plots that are not directly comparable in magnitude. However, the trends themselves highlight that the model's ability to generalize to the validation set remained a challenge throughout training.

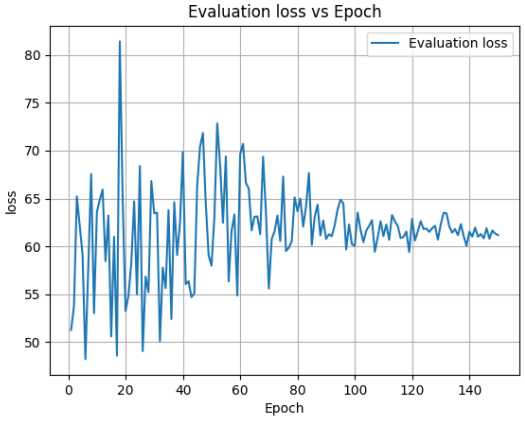


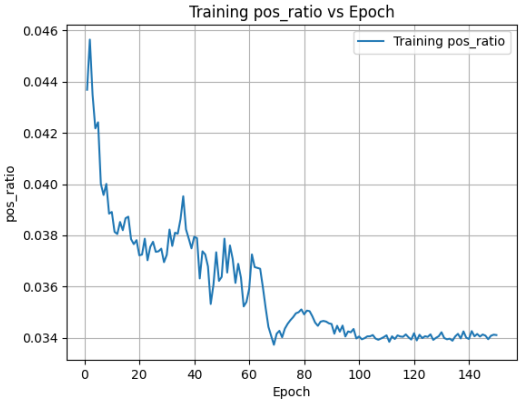
Figure 6.2: Evaluation loss over 150 epochs.

6.3.2 Positive and Negative Ratio Analysis

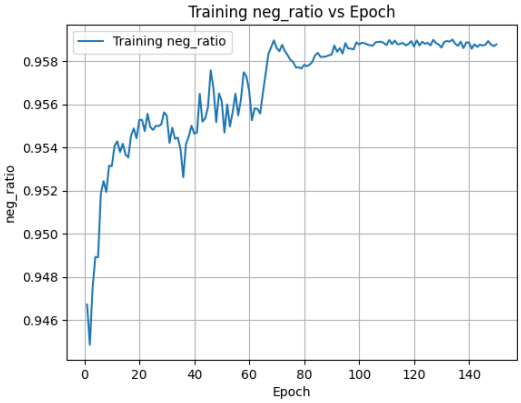
The positive and negative ratio metrics during training and evaluation provide insight into how the model aligns its classification with the dataset's object-to-background ratio of approximately 10% objects and 90% background points. Figures 6.3a, 6.3b, 6.3c, and 6.3d illustrate these trends.

The **training positive ratio** began relatively high and decreased steadily, stabilizing at around 0.034, which is slightly below the expected value. Similarly, the **evaluation positive ratio** showed an initial fluctuation but converged near 0.037, reflecting consistent underestimation of object points.

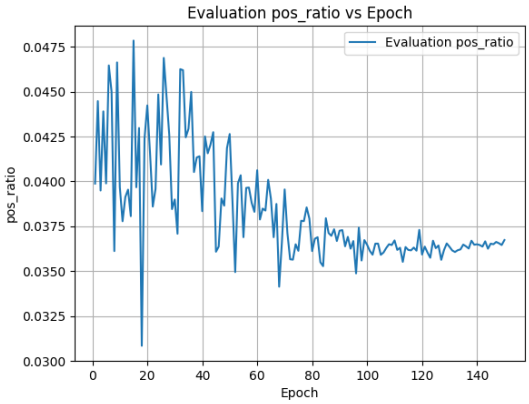
Conversely, the **training negative ratio** increased steadily, stabilizing near 0.958, closely matching the dataset's background point ratio. The **evaluation negative ratio** displayed a similar trend, converging near 0.955, indicating the model's generalization across unseen data.



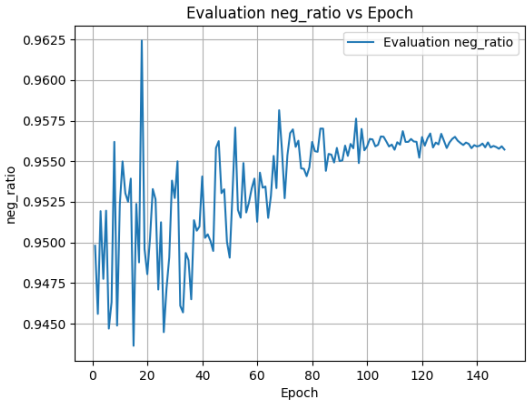
(a) Training Positive Ratio



(b) Training Negative Ratio



(c) Evaluation Positive Ratio



(d) Evaluation Negative Ratio

Figure 6.3: Training and evaluation positive and negative ratios: (a) Training Positive Ratio, (b) Training Negative Ratio, (c) Evaluation Positive Ratio, (d) Evaluation Negative Ratio.

6.3.3 Objectness Loss Analysis

Figure 6.4 illustrates the objectness loss during training. The **training objectness loss** started high but rapidly decreased within the first 20 epochs, followed by a slower decline and eventual stabilization around 0.04. This rapid decline and early plateauing could be attributed to the changes made to the loss calculation, where the objectness loss weights were increased to four times their original value. This adjustment amplified the influence of objectness loss during

optimization, leading to a sharper initial improvement in object classification.

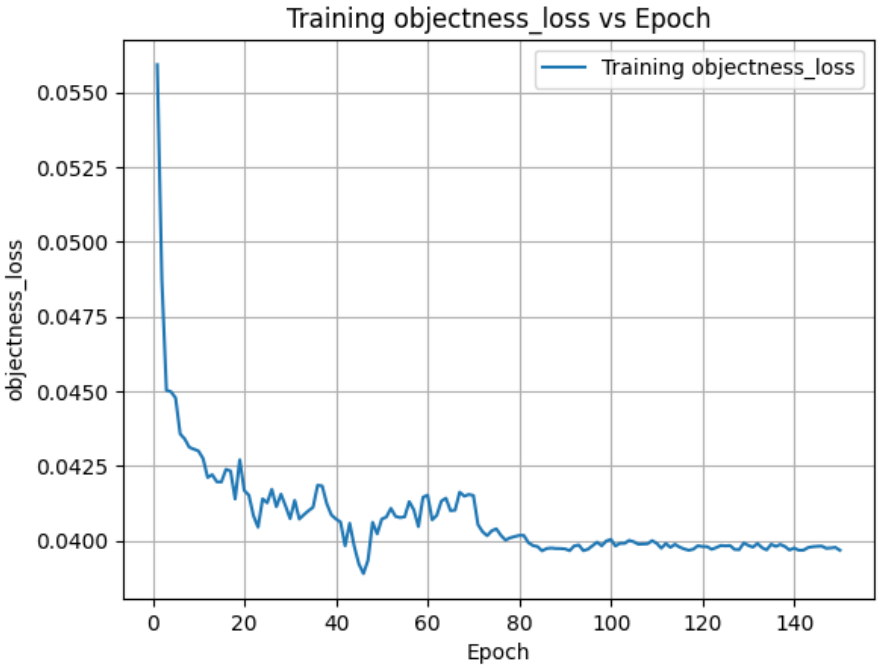


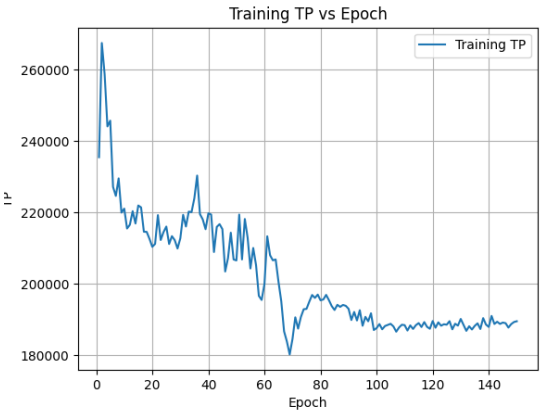
Figure 6.4: Training objectness loss over 150 epochs.

6.3.4 Confusion Matrix Parameters

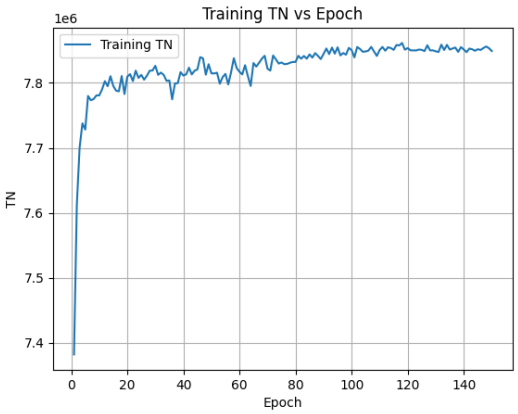
The confusion matrix parameters—True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN)—provide deeper insights into the performance of the model during training and evaluation. Figures 6.5 and 6.6 present the trends of these parameters over the epochs for both the training and evaluation processes.

Training Confusion Matrix The trends in training confusion matrix parameters are as follows:

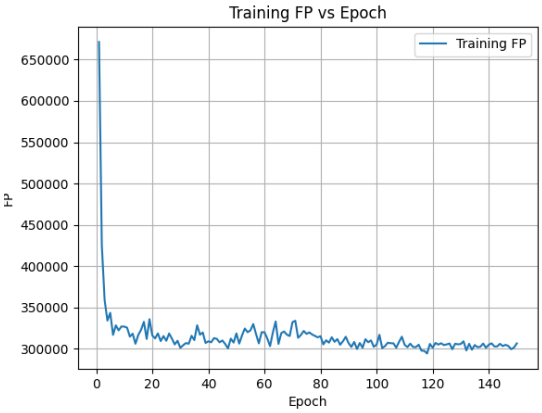
- **True Positives (TP):** A gradual decline was observed, stabilizing toward the later epochs. This suggests a consistent decrease in correctly identified object points, potentially due to the balancing of object and background classification during training.
- **True Negatives (TN):** TN steadily increased and plateaued, indicating that the model became proficient at correctly identifying background points as training progressed.
- **False Positives (FP):** FP exhibited a steep decline initially, followed by fluctuations and eventual stabilization, showing that the model improved its ability to avoid false classifications of background points as objects.
- **False Negatives (FN):** FN reduced significantly in the initial epochs but plateaued, reflecting some challenges in detecting all object points.



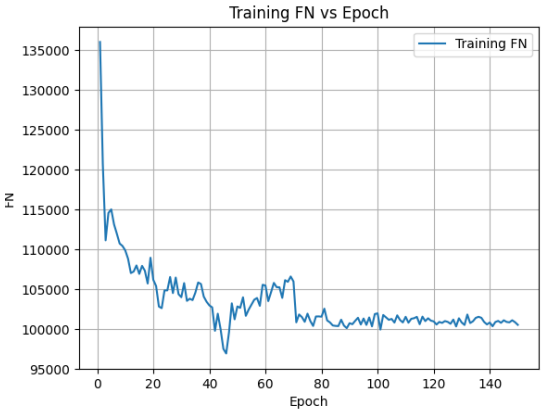
(a) Training True Positives (TP) vs Epoch



(b) Training True Negatives (TN) vs Epoch



(c) Training False Positives (FP) vs Epoch

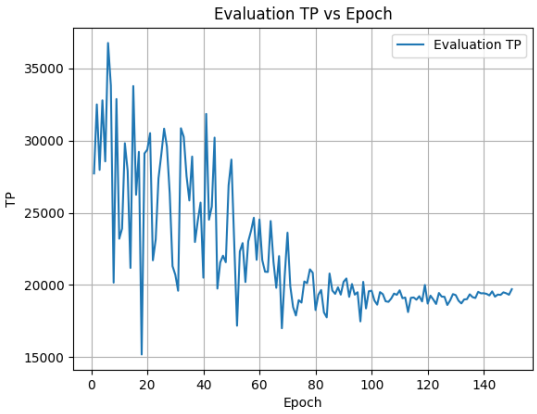


(d) Training False Negatives (FN) vs Epoch

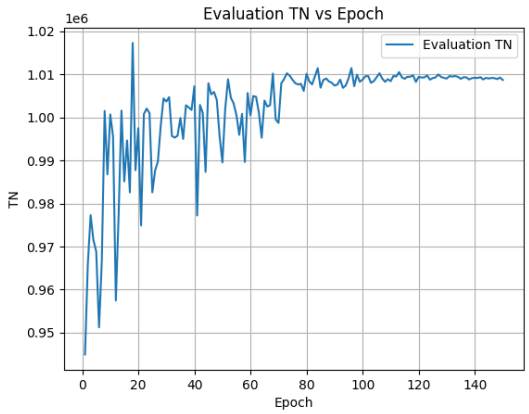
Figure 6.5: Confusion Matrix Parameters during Training: (a) Training True Positives (TP) vs Epoch, (b) Training True Negatives (TN) vs Epoch, (c) Training False Positives (FP) vs Epoch, (d) Training False Negatives (FN) vs Epoch.

Evaluation Confusion Matrix For the evaluation dataset:

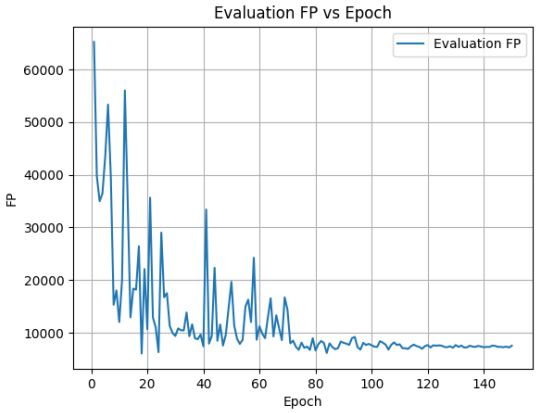
- **True Positives (TP):** TP remained relatively unstable initially and then plateaued, reflecting challenges in consistent object identification on unseen data.
- **True Negatives (TN):** Similar to training, TN increased steadily and stabilized, demonstrating robust performance in background point classification.
- **False Positives (FP):** FP showed significant fluctuations in the earlier epochs but gradually stabilized, indicating improvements in avoiding incorrect classifications.
- **False Negatives (FN):** FN exhibited significant variability initially before stabilizing, suggesting room for improvement in object detection.



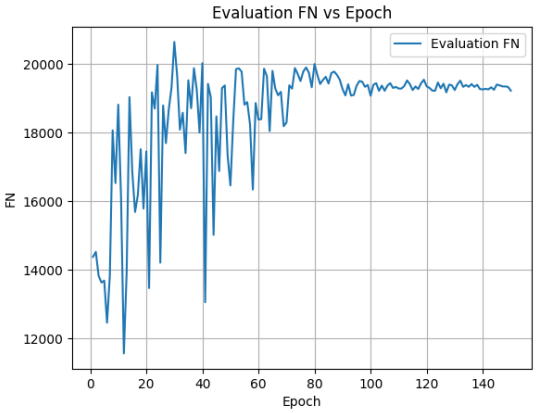
(a) Evaluation True Positives (TP) vs Epoch



(b) Evaluation True Negatives (TN) vs Epoch



(c) Evaluation False Positives (FP) vs Epoch



(d) Evaluation False Negatives (FN) vs Epoch

Figure 6.6: Confusion Matrix Parameters during Evaluation: (a) Evaluation True Positives (TP) vs Epoch, (b) Evaluation True Negatives (TN) vs Epoch, (c) Evaluation False Positives (FP) vs Epoch, (d) Evaluation False Negatives (FN) vs Epoch.

7 Examples and Results

This section presents the outcomes of the trained VoteNet model on the custom outdoor dataset, highlighting its ability to detect and classify objects under various scenarios. Through visualizations and quantitative metrics, the examples demonstrate the model's strengths in object detection as well as its limitations in challenging cases.

7.1 Examples

To demonstrate the model's capabilities, a dedicated inference script was developed. This script performs a forward pass on the trained model using point clouds from the test set, which is entirely separate from the training and evaluation datasets. The script utilizes Open3D to visualize the results, providing clear, color-coded bounding boxes for detected objects.

The color coding used in the visualizations is as follows:

- **Red**: Objects classified as "Benchy".
- **Blue**: Objects classified as "F1 Car".
- **Yellow**: Objects classified as "Coffee Mug".
- **Green**: Objects classified as "Rubber Duck".
- **Orange**: Objects classified as "Shuttle".

Below are selected examples from the test dataset, showcasing the model's performance in various scenarios:

7.1.1 Best Case Scenario

In this example (Figure 7.1), the model demonstrates excellent performance, successfully detecting and classifying all objects in the scene. This result highlights the model's capability when applied to well-defined, unobstructed objects.

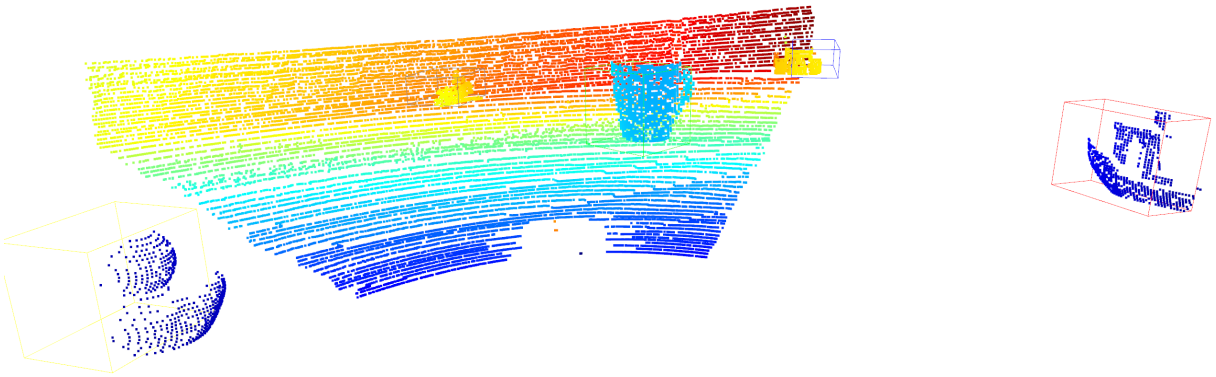


Figure 7.1: Best Case Scenario: All objects detected and classified correctly.

7.1.2 Adequate Detection

Figure 7.2 shows a scenario where the model performs adequately, identifying most objects but missing finer details. This demonstrates typical performance under standard conditions.

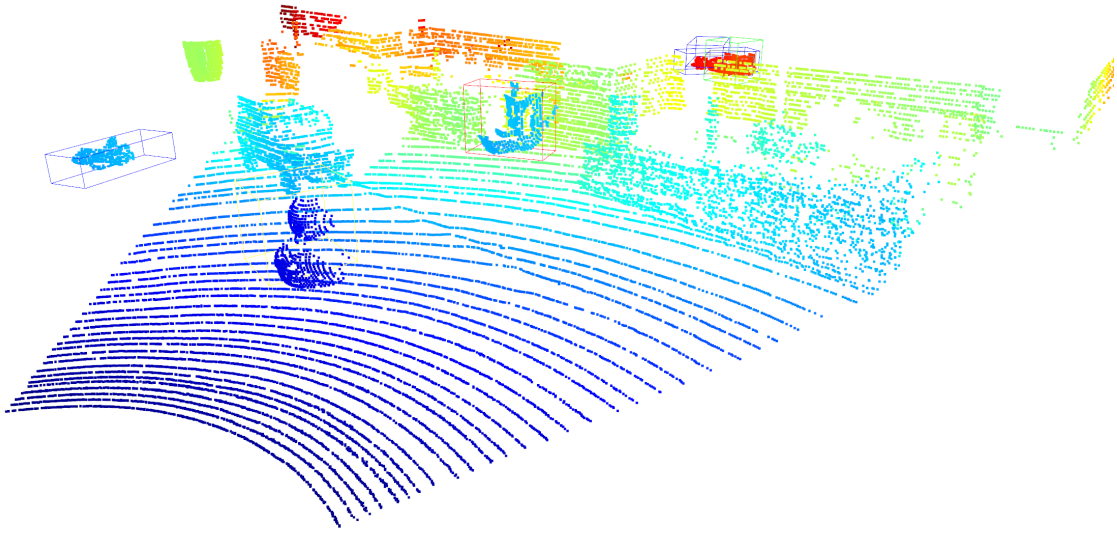


Figure 7.2: Adequate Detection: Most objects detected, though some details are missed.

7.1.3 Missed Objects

Figure 7.3 illustrates a situation where the model misses most of the objects in the scene. Such instances highlight areas where further improvements in model robustness or training dataset diversity may be required.

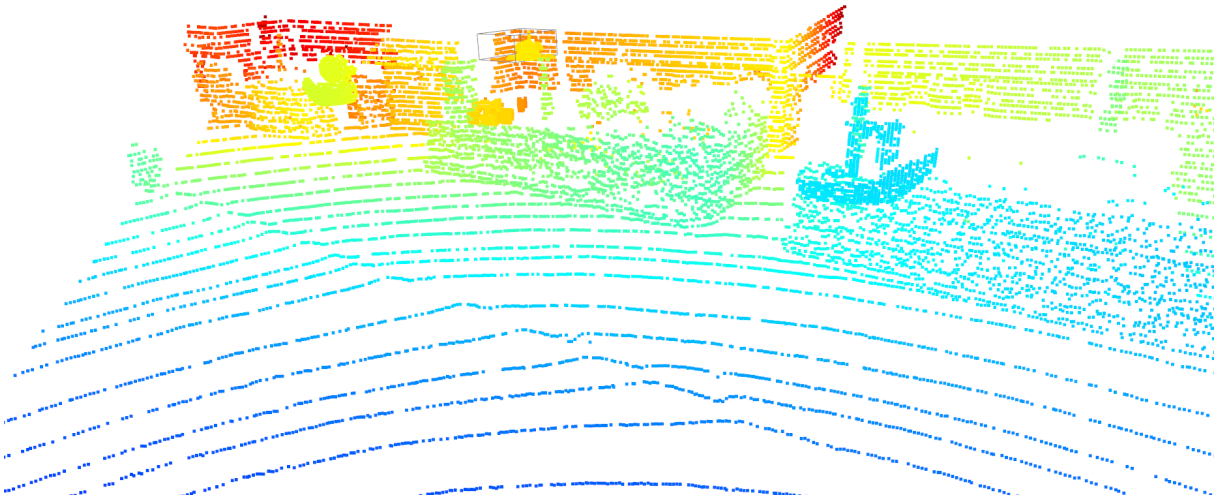


Figure 7.3: Missed Objects: Most objects not detected.

7.1.4 False Positives and Ambiguities

In certain cases, the model generates multiple classifications for a single object, as shown in Figure 7.4. This indicates ambiguity in object recognition, particularly in scenes with overlapping or closely packed objects.

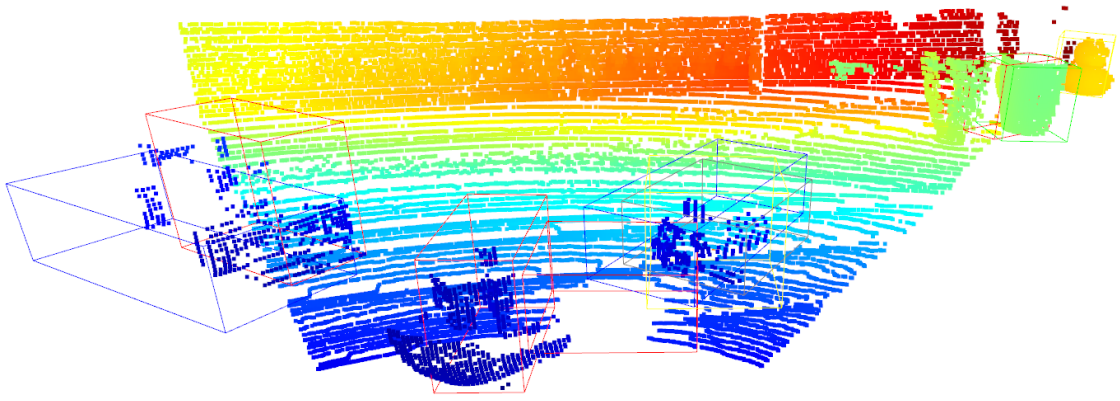


Figure 7.4: Ambiguous Classification: Multiple detections for a single object.

7.1.5 Clean Scene with No Misclassification

Finally, in Figure 7.5, the model processes a scene with no objects misclassified, accurately recognizing the absence of objects in certain regions.

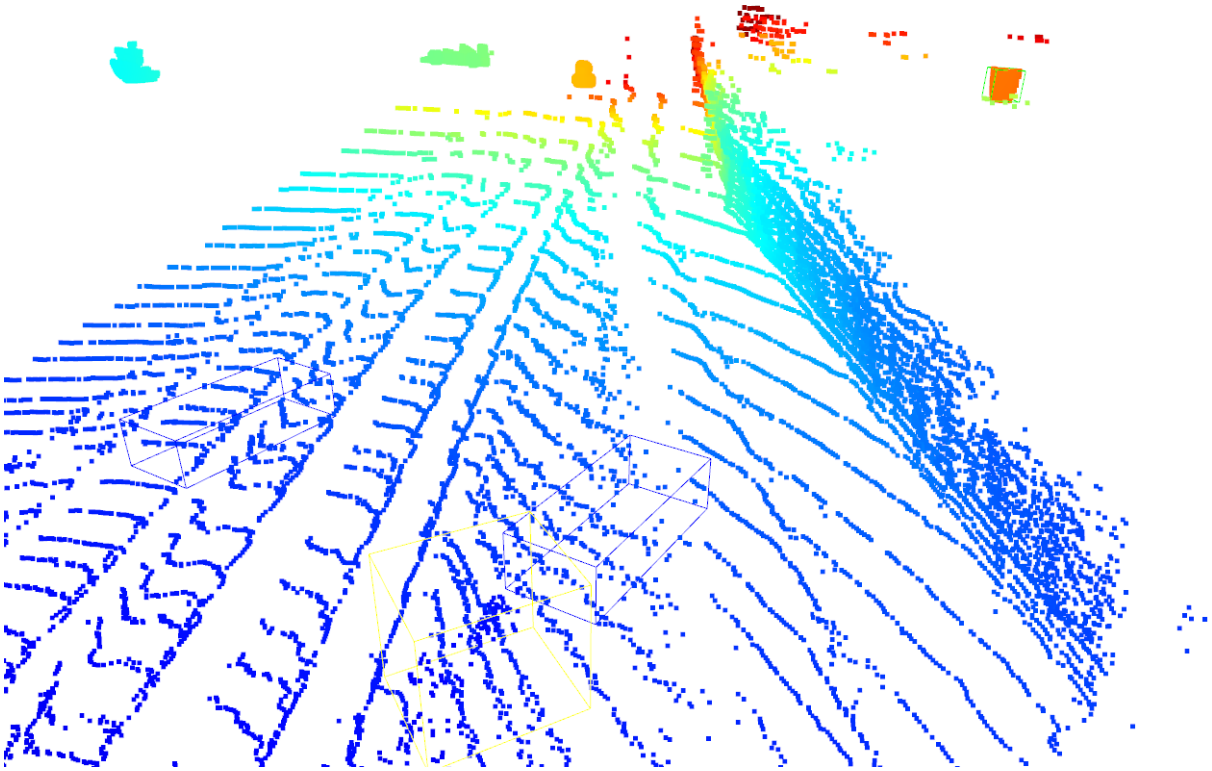


Figure 7.5: No Misclassification: Clean detection without false positives.

7.2 Results

To evaluate the model's performance, an evaluation script was developed. This script processes the test dataset, distinct from both training and evaluation sets, ensuring an unbiased assessment of the model's generalization capabilities. The script computes performance metrics, including loss components, precision, recall, confusion matrix elements, and class-specific average precision (AP). Below, we summarize the key results obtained from the evaluation.

7.2.1 Overall Metrics

The evaluation results for overall metrics are presented in Table 3.

Table 3: Overall Evaluation Metrics

Metric	**Value**
Mean Loss	48.16
Center Loss	0.94
Objectness Loss	0.043
Heading Classification Loss	2.01
Heading Regression Loss	0.14
Size Classification Loss	0.89
Size Regression Loss	0.0013
Semantic Classification Loss	0.89
Average Positive Ratio	0.046
Average Negative Ratio	0.945
Precision	40.08%
Recall	74.48%

7.2.2 Confusion Matrix

The confusion matrix results are summarized in Table 4.

Table 4: Confusion Matrix Results

Metric	**Value**
True Positives (TP)	36,686
False Positives (FP)	54,852
True Negatives (TN)	949,778
False Negatives (FN)	12,570

7.2.3 Class-Specific Performance

Table 5 presents the class-specific performance metrics, including Average Precision (AP) and Recall for each object class.

Table 5: Class-Specific Performance Metrics

Class	**Average Precision (AP)**	**Recall (%)**
Benchy	0.0051	24.70
Coffee Mug	0.3213	64.80
F1 Car	0.0082	7.94
Rubber Duck	0.1771	54.54
Shuttle	0.1542	28.54

The **mean Average Precision (mAP)** across all classes is **0.1332**, and the **Average Recall (AR)** is **36.10%**. These results demonstrate variability in the model's performance across different object classes, with better detection performance for objects like "Coffee Mug" and "Rubber Duck" and challenges with objects like "Benchy" and "F1 Car."

8 Challenges and Mitigation

The implementation of the VoteNet model for this project required addressing several challenges, primarily related to adapting the model for a custom dataset, addressing potential issues with the model's implementation, and generating a compatible dataset. Both tasks—model adaptation and dataset generation—proved to be significantly more complex and larger in scope than originally anticipated.

8.1 Adaptation of the VoteNet Model

Although the VoteNet repository included a readme on training the model with custom datasets, its overall design was not optimized for this flexibility. The training script, in particular, was closely tied to the two datasets from the original paper (SUN RGB-D and ScanNet), making it challenging to adapt for new datasets. This process turned out to be far more time-intensive and complex than initially expected.

One unexpected challenge was the need to modify parameters in the loss calculation script to align with the new dataset. This critical step, which addressed differences in the distribution of objects and background points, was not mentioned in the documentation. The omission added another layer of complexity to an already demanding task. Without clear instructions or a modular framework, adapting the repository proved significantly more challenging than anticipated.

8.2 Reported Issues with the Model

During the project, pretrained weights provided in the repository were tested on a single scene using the demo script. These weights produced results as expected for the given scene, demonstrating that the original implementation works under specific conditions. However, since I did not perform a full evaluation of the original implementation or retrain the model on the datasets used in the paper, I cannot definitively state whether the model reproduces the results reported in the original paper.

That said, while researching common issues encountered by others using VoteNet, I observed multiple reports of difficulties with training the model on custom datasets. Some users described challenges with models classifying all points as background, an issue similar to the one encountered and solved in this project. Others reported difficulties reproducing the original results. Some community members speculated that minor discrepancies might exist between the published model in the paper and the code provided in the repository, particularly in the forward pass algorithm. Although these observations cannot be confirmed without further testing, they highlight potential areas for improvement in the model's implementation and documentation.

8.3 Dataset Generation Complexity

Generating a dataset compatible with VoteNet was another significant challenge in this project. The process went beyond simply collecting point clouds and annotations, requiring creative solutions to simulate realistic LiDAR data and ensure compatibility with the model. This task also proved to be significantly more complex and time-consuming than initially expected, as described below:

- **Simulation of LiDAR Scans:** Creating point clouds for custom objects involved simulating LiDAR-like data using a Z-depth approach. This method enabled the generation of 3D point clouds for objects not present in standard datasets. Implementing this process required substantial experimentation and refinement to ensure the simulated scans were realistic and consistent with actual LiDAR data.

- **Vote Generation and Annotation:** The custom dataset required a script to calculate ground truth votes and annotations for each object in a scene. This script needed to handle both the geometric properties of the objects and the spatial layout of the scenes, ensuring accurate data representation. Developing this script demanded a deep understanding of the VoteNet architecture and the properties of 3D datasets, making it a significantly larger task than initially estimated.
- **Data Configuration File:** To streamline the integration of the custom dataset, a `data_config` file was developed. While not a requirement in the original implementation, this file centralized dataset-specific parameters, making it easier to manage and adapt the training process. This addition significantly improved modularity and reduced the risk of configuration errors, but it required additional effort to design and implement effectively.

9 Summary

9.1 Conclusions

The results of this project provide several key insights into the performance and limitations of the VoteNet model when applied to outdoor scenes:

1. **Challenges in Achieving Reliable Outdoor Performance:** This project evaluated the VoteNet model's suitability for outdoor environments by training, evaluating, and testing it on partial scans designed to simulate real-world scenarios. In contrast, the original VoteNet paper used datasets comprising full scans of entire scenes, providing a more comprehensive view of the environment. This difference likely contributed to the observed challenges in achieving reliable detection performance, as the reduced context in partial scans limits the model's ability to interpret scene structure and detect objects effectively.

2. **Impact of Training Parameters:** The testing results highlighted the importance of careful calibration of the training parameters, such as loss function weights, in determining the model's performance. The suboptimal balance between objectness, semantic classification, and other loss components may have contributed to the low precision observed during evaluation. Adjustments to these weights could improve the model's ability to distinguish objects from the background in outdoor scenes.

3. **Effect of Reduced Point Cloud Density:** The project utilized point clouds with a maximum of 20,000 points per scene, constrained by the characteristics of the KITTI dataset. This contrasts with the 40,000 points used in the original VoteNet experiments. The lower point density likely impacted the model's ability to capture fine details, leading to reduced classification accuracy for certain object types.

4. **Potential Overfitting:** The significant gap between training and evaluation metrics suggests potential overfitting during the training process. This is supported by the oscillatory nature of the evaluation loss and the disparity in performance between the training and test datasets. One possible contributor to this overfitting could be an imbalance in the loss function weights, which may have caused the model to overly prioritize certain loss components at the expense of others. Further regularization and data augmentation strategies could help mitigate this issue.

5. **Class-Specific Performance Variability:** The model demonstrated inconsistent performance across object classes, with objects like *Coffee Mug* achieving relatively high average precision, while others like *Benchy* and *F1 Car* performed poorly. One potential explanation for this disparity is the geometric simplicity and symmetry of the *Coffee Mug* (excluding the handle), which likely allowed it to retain a relatively consistent appearance after being processed by the LiDAR simulation. In contrast, more complex objects such as the *Shuttle* and *F1 Car* may have experienced greater variability in their projections, making them harder to classify accurately.

6. **Generalization Challenges:** Despite extensive modifications to adapt the model for outdoor datasets, the evaluation metrics indicate that the model struggles to generalize effectively to outdoor environments. This highlights the need for more diverse and representative datasets, as well as algorithmic enhancements, to bridge the gap between indoor and outdoor object detection tasks.

9.2 Future Research

While this project made significant strides in adapting and testing the VoteNet model for outdoor object detection, several avenues remain open for future research and development:

1. **Fine-Tuning Training Parameters:** Due to time and resource constraints, this project included only two full training runs on the custom dataset. As a result, there is substantial potential for improving model performance by fine-tuning training parameters, such as the learning rate, batch size, and loss function weights. A more extensive hyperparameter search could yield configurations that better align with the characteristics of the custom dataset.

2. **Increased Point Cloud Density:** This project utilized scenes with a maximum of 20,000 points per scan due to the constraints of the KITTI dataset. Future work could explore training and testing the model on datasets with higher point densities, such as 40,000 points per scene, to evaluate whether the reduced point cloud resolution contributed to the observed performance limitations.

3. **Alternative Models:** Given the challenges encountered with the VoteNet model in this project, exploring alternative 3D object detection architectures may be worthwhile. Models specifically designed for outdoor environments or that incorporate techniques such as transformer-based architectures, voxel-based representations, or multi-view approaches may offer improved performance.

4. **Improving Data Generation:**

   (a) **Refining Object Placement Boundaries:** In the current data generation process, objects are placed within the horizontal bounding box defined by the maximum points in the scene. However, this bounding box does not accurately reflect the triangular field-of-view (FOV) of the simulated LiDAR scans, leading to potential placement of objects outside the FOV. Future improvements could implement placement constraints that better align with the actual FOV of the scene.

   (b) **Simulating Occlusions:** Currently, the object placement process does not account for occlusions. Objects are rendered as if fully visible, even when positioned behind other objects or walls relative to the camera origin. Adding an occlusion simulation step to the data generation process could yield more realistic datasets and enhance model robustness.

   (c) **Adding Noise to Simulated Scans:** The simulated LiDAR scans for the placed objects are noise-free in this project. Introducing noise into the dataset, similar to real-world LiDAR data, could help the model generalize better to practical scenarios.

5. **Reproducing and Validating Results:** Finally, reproducing the VoteNet results on its original datasets and testing its performance in outdoor scenes with extensive hyperparameter tuning could clarify its applicability and limitations. Validating the findings in this project with larger-scale experiments could provide additional insights.

References

[1] Texas Instruments. "Time-of-Flight Cameras". In: *TI Whitepaper* (2023). URL: https://www.ti.com/lit/wp/sloa190b/sloa190b.pdf.

[2] Springer. "Accuracy Challenges of Time-of-Flight Cameras". In: *Machine Vision and Applications* (2016). URL: https://link.springer.com/article/10.1007/s00138-016-0784-4.

[3] Artec 3D. *Structured Light 3D Scanning*. 2023. URL: https://www.artec3d.com/learning-center/structured-light-3d-scanning.

[4] 3D Insider. *Structured Light 3D Scanning*. 2023. URL: https://3dinsider.com/structured-light-3d-scanning/.

[5] Vision Center. *What is Stereoscopic Vision?* 2023. URL: https://www.visioncenter.org/conditions/stereoscopic-vision/.

[6] ClearView Imaging. *Stereo Vision for 3D Machine Vision Applications*. 2023. URL: https://www.clearview-imaging.com/en/blog/stereo-vision-for-3d-machine-vision-applications.

[7] University of Waterloo. "Exploring RGB-D Data Representations for Augmented Reality and Robotics". In: *ISPRS Archives* (2019). URL: https://uwaterloo.ca/geospatial-intelligence/sites/default/files/uploads/files/isprs-archives-xlii-2-w13-835-2019.pdf.

[8] Springer. "Efficient Processing of 3D Point Clouds". In: *The Visual Computer* (2022). URL: https://link.springer.com/article/10.1007/s00371-022-02661-5.

[9] Springer. "Voxel-Based Representations for 3D Medical Imaging". In: *Journal of Supercomputing* (2021). URL: https://link.springer.com/content/pdf/10.1007/s11227-021-03899-x.pdf.

[10] Springer. "Mesh Models in Computer Graphics: A Comprehensive Review". In: *Artificial Intelligence Review* (2024). URL: https://link.springer.com/article/10.1007/s10462-024-10941-w.

[11] Charles R Qi et al. "Deep Hough Voting for 3D Object Detection in Point Clouds". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9277–9286.

[12] Charles R Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *arXiv preprint arXiv:1706.02413* (2017).

[13] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. "Real-Time Seamless Single Shot 6D Object Pose Prediction". In: *CVPR*. 2018.

[14] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* (2013).

[15] Shuran Song, Shane Lichtenberg, and Jianxiong Xiao. "SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 567–576.

[16] Nathan Silberman et al. "Indoor Segmentation and Support Inference from RGBD Images". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2012, pp. 746–760.

[17] Angela Dai et al. "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5828–5839.

[18] R Parikh et al. "Understanding and using sensitivity, specificity and predictive values". In: *Indian journal of ophthalmology* 56.1 (2008), pp. 45–50. DOI: 10.4103/0301-4738.37595.

[19] Takaya Saito and Marc Rehmsmeier. "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets". In: *PLOS ONE* 10.3 (2015), e0118432. DOI: 10.1371/journal.pone.0118432.

[20] James D. Foley et al. *Computer Graphics: Principles and Practice*. 2nd. Reading, MA: Addison-Wesley, 1995. ISBN: 978-0201848403.