Point Cloud Based 3D Object Detection Using VoxelNet with RoI Pooling

Anbarasan Kandasamy
University of Maryland
College Park, MD, USA
anbuk@umd.edu

Hamsaavarthan Ravichandar
University of Maryland
College Park, MD, USA
rhamsaa@umd.edu

Manoj Kumar Selvaraj
University of Maryland
College Park, MD, USA
manojs@umd.edu

***Abstract*—Detection of surrounding objects in a vehicle is the most crucial step in autonomous driving. Failure to identify those objects correctly in a timely manner can cause irreparable damage, impacting our safety and society. The 2D object detection method has achieved remarkable success; however, in recent years, object detection in 3D has received more remarkable adoption [1], especially when dealing with the sparsity and unstructured property of point-clouds. This paper discusses point-cloud-based 3D object detection using VoxelNet: a generic 3D detection network that unifies feature extraction and bounding box prediction into a single stage, end-to-end trainable deep network, incorporating an enhanced neural network. VoxelNet tackles the challenge of accurate 3D object detection, which is critical for autonomous systems, robotics, and augmented reality. Traditional methods rely on manual feature extraction, limiting the use of 3D shape data from LiDAR point clouds. VoxelNet overcomes this by introducing an end-to-end deep learning architecture that voxelized point clouds, eliminating the need for handcrafted features. This approach enables more effective learning of expressive and discriminative features, significantly improving the detection of objects such as cars, pedestrians, and cyclists, and advancing the performance, scalability, and real-time capabilities of autonomous systems. In light of this perspective, we try to adopt a straightforward yet powerful voxel-based framework called Voxel R-CNN [10], where we implement Voxel RoI pooling to extract RoI features, which are fed into the detect subnet for box refinement. At a fraction of the computation cost, our method achieves detection accuracy comparable to state-of-the-art point-based models by fully utilizing voxel features in a two-stage approach.**

I. Introduction

3D object recognition has several advantages over 2D detection methods, as more accurate information about the environment is obtained for better detection. For example, the depth of the images is not considered in 2D detection, which reduces the detection accuracy. The development of autonomous systems, particularly in 3D object detection, is substantially enhanced by using LiDAR point clouds. Current developments in 3D object detection mainly depend on the representation of the 3D data, whether it be point-based or voxel-based. Because this structure can better store accurate point positions, a large number of high performance 3D detectors currently in use are point-based. However, because of their unordered storage, point-level characteristics result in significant computing overheads. On the other hand, because the input data is separated into grids, the voxel-based structure is more appropriate for feature extraction but frequently produces inferior precision.

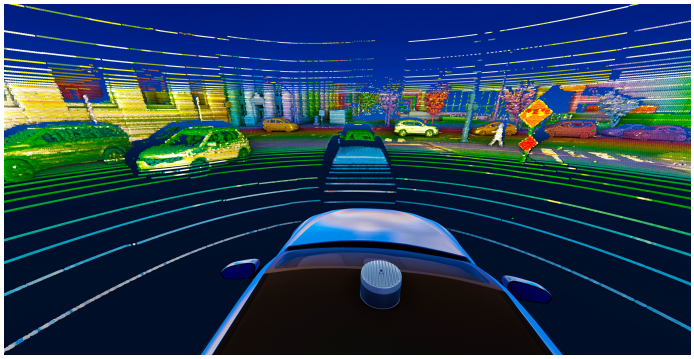

Fig. 1. 3D Point Cloud Image Annotation Based on LiDAR Sensors [2]

Traditional approaches are exhaustive and rely heavily on manual feature engineering, which can limit the efficacy of point-cloud data in accurately detecting objects. This project aims to implement a feature learning network for an end-to-end, trainable 3D object detection framework based on VoxelNet.

A. Related Work

Relevant works on 3D detection methods currently in existence can be divided into two groups: point-based and voxel-based. Due to their superior memory locality, the voxel-based methods [11] divide point clouds into regular grids, which are more suitable for convolutional neural networks (CNNs) and more effective for feature extraction. However, voxelization frequently results in the loss of precise position information, which is a drawback. The majority of the most advanced 3D detectors available today are point-based, abstracting a set of point representations through iterative sampling and grouping after receiving raw point clouds as input [12] [13]. On a number of benchmarks, the sophisticated point-based methods [14] rank highest. As a result, it is now widely believed that accurate object localization depends on the precise position information found in raw point clouds. Despite the superior detection accuracy, point-based methods are in general less efficient because it is more costly to search nearest neighbor with the point representation for point set abstraction. Additionally,

a number of multi-modal fusion techniques combine LiDAR and images to increase detection accuracy [5] [15]. Since cameras provide an order of magnitude more measurements than LiDAR, these techniques perform better than LiDAR-only 3D detection, especially for small objects (bikes, pedestrians) or when the objects are far away.

B. Our Approach

- In this project, our approach was builded on the foundation for automatic feature extraction from point-cloud data, enabling the model to learn the best features for detection without extensive manual design.
- Taking inspiration from the Voxel R-CNN, that consists of a 3D backbone network, a 2D bird-eye-view (BEV) Region Proposal Network and a detect head. But for this project, we only take full advantage of VoxelNet 3D detection network features and try to implement a Region of Interest (RoI) layer following the final Region Proposal Network (RPN).
- Our method achieves comparable detection accuracy with state-of-the-art manual feature extraction models, but at a fraction of the computation cost. A voxel RoI pooling is devised to extract RoI features directly from voxel features for further refinement.

II. Objective

The core objective of this project is to develop a robust feature learning network for 3D object detection in LiDAR point cloud data. Our approach leverages voxel partitioning to transform raw point cloud inputs into a structured feature representation, aiming to facilitate the subsequent stages of detection. The problem involves managing and preprocessing large amounts of.bin files that represent training data. Efficiently loading, processing, and storing batches of data is critical to ensure smooth and accelerated training.

A. Model Datasets

The dataset that we use is the KITTI 3D Object Detection Benchmark Datasets, which provides LiDAR point cloud data and ground truth labels for multiple object classes (e.g. cars, pedestrians, cyclists). The KITTI dataset offers annotated data for tasks such as 3D object detection, visual odometry, scene flow estimation, and stereo vision. The car dataset, in particular, focuses on vehicle detection and localization in urban settings. The KITTI dataset consists of 7481 training images and 7518 test images as well as the corresponding point clouds, in total of around 80000 labeled data.

B. Model Input/Output

The input is a point cloud divided into voxels, each containing a set of points with coordinates and reflectance values. Using a LiDAR sensor, data is collected and represented as a point cloud of individual points reflected from the surface of objects in a 3D environment. The output of the feature learning network is a set of voxel-wise feature representations, which are later used by a Region Proposal Network (RPN) to predict bounding boxes around detected objects.

C. Evaluation Metrics

Intersection over Union (IoU) is a metric used to evaluate the performance of object detection models. It measures the overlap between the predicted bounding box and the ground truth bounding box. IoU is defined as the ratio of the area of overlap to the area of union:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{1}$$

Where:

- **Area of Overlap**: The region where the predicted and ground truth bounding boxes intersect.
- **Area of Union**: The total area covered by both the predicted and ground truth bounding boxes.

For car detection, KITTI adopts an IoU threshold of **0.7**:

- A detection is considered positive if its IoU with the ground truth box exceeds 0.7.
- Predictions with IoU between 0.45 and 0.6 are categorized as don't care, meaning they are ignored during evaluation.
- Predictions with IoU less than 0.45 are treated as false positives.

D. Expected Results

We anticipate an improvement over the baseline models that rely on manually designed features using learned feature representations. This should lead to higher accuracy in 3D bounding box prediction, especially under more complex scenarios with sparse point data, using the idea of end-to-end LiDAR-based learning instead of using hand-crafted features.

E. Difference From the State-of-the-Art

We synchronize the LiDAR and image data to combine spatial and visual information, providing additional parameters for more detailed 3D object representation. Our model learns voxel-based features directly from point cloud data, a more scalable approach that integrates directly with the modified faster R-CNN model called the Regional Proposal Network (RPN).

III. Methodology

VoxelNet is a deep learning architecture designed for 3D object detection, especially in applications like autonomous driving and LiDAR data analysis. It transforms raw point cloud data, typically collected from LiDAR sensors, into a structured voxel grid, allowing for efficient processing through 3D convolutional neural networks (CNNs). The VoxelNet framework, introduced in the original paper [3], contains three main components: a feature learning network, convolutional middle layers, and a region proposal network (RPN). The main contributiona and the novel approaches of this paper will be implemented in the upcoming Voxel Feature Encoding (VFE) layer and the modified faster R-CNN, Regional Proposal Network (RPN) layer.

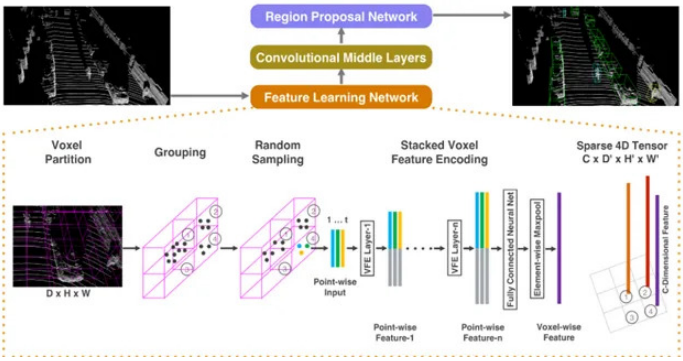

Fig. 2. VoxelNet Architecture [3]

A. Data Preprocessing

Point clouds obtained using LiDAR are sparse and can contain greater than 100k points; therefore, using all the information at once can be computationally expensive. However, VoxelNet divides the 3D space into equally spaced voxels, each with a specified dimension. Each voxel will now contain a variable number of points due to the sparsity of LiDAR data. To overcome this, we use random sampling to sample a fixed number of points, T, from each voxel prior to encoding the information via the Voxel Feature Encoding (VFE) layer which is nothing but a Feature Learning Network. The random sampling and coordinate lookup can be constructed in a single pass; therefore, the complexity of this operation is O(n). The number of points chosen is specific to the task. More points will allow for capturing better shape information; however, too many points would make the network computationally expensive. This approach provides an added benefit of variation in training data as well as reducing the imbalance in number of points per voxel.

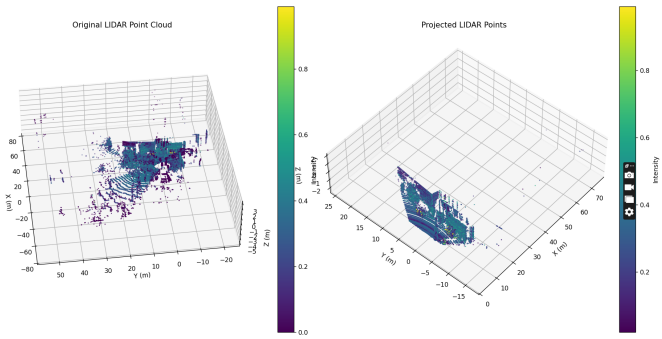

Fig. 3. Preprocessed LiDAR data w.r.t camera frame

B. Feature Learning Network

The feature learning network consists of Voxel Feature Encoding (VFE) layers, which enable inter-point interaction within each voxel. Each point is passed through a fully connected neural (FCN) network, and local features are aggregated using max-pooling to obtain a single, descriptive feature per voxel. This output is represented as a sparse 4D tensor, reducing memory and computational load for subsequent network layers. The VFE layer is what set apart VoxelNet from previous work and introduced the idea of end-to-end learning instead of using hand-crafted features. These hand-crafted features were often computed using certain characteristics of the data.

C. Convolutional Middle Layers

After the features of each voxel are encoded using the VFE layer, they're processed further through the Convolutional Middle Layers [6] [7]. Each of the layers is a 3D convolution with a batch normalization layer and a ReLU layer. The goal of these layers is to aggregate voxel-wise features while progressively expanding the receptive field. The figure below shows an outline of the Convolutional Middle Layers architecture for Car Detection using the KITTI dataset.

Fig. 4. Convolutional Middle Layers for Car Detection [4]

D. Regional Proposal Network (RPN)

To generate 3D detections, the features from the Convolutional Middle Layers are passed to a Region Proposal Network. Prior to passing the output of the Convolutional Middle Layers to the RPN the features must be reshaped to a 3D tensor. After reshaping, the RPN uses convolutions to down sample the features which are passed using skip connections then up sampled to a fixed size for the final feature map. The final feature map is used to predict a probability score map and a regression map. The network provides an output for each voxel. For 2D images, there exists a popular state of the art network, such as Faster R-CNN, that has implemented RPN's for object detection using 2D images. The authors of the original VoxelNet, adapt the same RPN used in Faster R-CNN for 3D object detection [5]. However, the novelty of this paper is quartered in the unique and enhanced implementation of the RPN layer for 3D object detection, which will be discussed in the further implementation of the project [8] [9].

E. Going above and beyond with Region of Interest (RoI) Pooling Layer

Integrating an RoI Pooling layer into the VoxelNet architecture would involve modifying its single-stage pipeline to include a two-stage detection process, similar to the approach used in Voxel R-CNN. The original Voxel R-CNN [10] implimentation includes: (a) a 3D backbone network, (b) a 2D backbone network followed by the Region Proposal Network (RPN), and (c) a voxel RoI pooling and a detect subnet for box refinement. But as informed, in this project we try to incoporate the RoI pooling layer from R-CNN in the orioginal VoxelNet architecture. The first-stage of the network has three

main components from the VoxelNet architecture as explained earlier:

- Voxel Feature Encoding (VFE) Layers: Encodes raw point cloud data into a voxel-wise representation.
- Convolutional Middle Layers: Applies 3D convolutions to extract global voxel features.
- Region Proposal Network (RPN): Generates 3D object proposals directly from the voxel features.

To integrate an RoI Pooling layer, you need to insert the RoI pooling layer after the Region Proposal Network (RPN) layer. This allows you to refine the object proposals by pooling features from the high-resolution feature maps. From the 3D bounding box proposals generated by the RPN of voxel feature maps, the implementation of second-stage of this network begins:

- Introduce an RoI Pooling module that extracts features from the voxel feature maps for each 3D proposal.
- The RoI Pooling layer aggregates features from the regions defined by the proposals.
- After pooling, a fully connected network or small CNN processes these pooled features for refined classification and regression.

Given the the RoI layer must take inputs from the RPN layer, therefore the RoI pooling layer should work in 3D feature space. The region proposals from RPN is first divided into KxKxK regular sub-voxels. The center point is assumed to be the associated sub-voxel's grid point. We are unable to directly use max pooling over features of each sub-voxel as in [17] because 3D feature volumes are very sparse (non-empty voxels account for $< 3\%$ spaces). Rather, in order to extract features, we incorporate features from nearby voxels into the grid points.The design implementation of the RoI layer is as follows:

- Input: The feature map from the 3D convolutional backbone and the 3D RoIs generated by the RPN.
- Grid Sampling: Divide each 3D RoI into a fixed grid of sub-regions.
- Feature Pooling: Extract features from the corresponding sub-regions in the voxel feature map using max pooling or average pooling.
- Output: A fixed-size feature vector for each RoI.

Finally, Add a refinement network layer after RoI pooling to obtain final output results (labels) as mentioned below:

- Feature Processing: Pass the pooled features through fully connected layers to further refine the classification/detection scores and bounding box coordinates.
- Final Outputs: Generate the final object detection and refined 3D bounding boxes.

IV. Experiments

A. *Ablation Study*

- **Method (a)** represents the *hand-crafted baseline*, which uses bird's-eye view (BEV) features with handcrafted statistical quantities to encode voxels. It achieves moderate AP of 77.32%, which demonstrates the limitations of hand-crafted features in representing the complex 3D spatial information.
- **Method (b)** replaces the handcrafted features with a *single-layer Voxel Feature Encoding (VFE) module*. This results in an increase of 3.8% in AP to 81.12%, indicating that learned voxel features are more discriminative for 3D detection tasks. However, it incurs additional computational cost due to the feature extraction process.
- **Method (c)** extends (b) by introducing *stacked VFE layers* to encode hierarchical shape information. This leads to a further improvement of 3.5% AP, reaching 84.62%, showing that deeper feature representation better captures the complex local geometry of objects.
- **Method (d)** integrates the *3D convolutional middle layers* to aggregate spatial context across voxels. This addition enhances moderate AP to 87.45%, demonstrating the importance of incorporating global spatial context for accurate object localization.
- **Method (e)** is the proposed *VoxelNet architecture*, which combines stacked VFE layers, 3D convolutional middle layers, and an RPN for region proposals. This complete model achieves 89.60% AP, a significant improvement over the baseline, and represents the state-of-the-art accuracy for LiDAR-based 3D object detection.

The results clearly demonstrate that each module contributes to both the accuracy and efficiency of the final model, and the integration of these components enables VoxelNet to perform robust 3D object detection directly on sparse LiDAR point clouds.

Table I details how each proposed module influences the accuracy and efficiency of VoxelNet. The results are evaluated with Average Precision (AP) on the KITTI validation set for the moderate level of the car detection task.

Method	Description	Moderate AP (%)
(a)	Hand-crafted BEV baseline	77.32
(b)	Single-layer VFE	81.12
(c)	Stacked VFE layers	84.62
(d)	Added 3D convolutional middle layers	87.45
(e)	VoxelNet	89.60

TABLE I
ABLATION STUDY RESULTS ON KITTI VALIDATION SET (MODERATE LEVEL AP FOR CAR CLASS)

B. *Hyperparameter Tuning*

1. Voxel Size

The voxel size was set to $vD = 0.4\,\text{m}, vH = 0.2\,\text{m}, vW = 0.2\,\text{m}$, where vD is the depth, vH is the height, and vW is the width of each voxel.This configuration ensures a balance between capturing fine-grained geometric details and computational efficiency, with smaller vH providing better height resolution critical for vehicle detection.

2. Maximum Points per Voxel

The maximum points per voxel (T) was set to $T = 40$. This value was chosen to capture sufficient local geometric information while maintaining computational efficiency. Increasing

T from lower values ensures that densely populated voxels retain more point-level details, which is critical for accurately representing larger and more complex objects like cars. At the same time, $T = 40$ avoids excessive computational overhead that would arise from processing too many points within each voxel, striking an optimal balance between accuracy and efficiency.

3. Learning Rate

The learning rate (lr) was set to 0.01 for the first 15 epochs. This value was chosen to ensure rapid convergence during the initial stages of training, allowing the model to quickly learn foundational patterns in the data. The relatively short duration of 15 epochs for this higher learning rate avoids overshooting the optimal solution, particularly in the early phases of optimization. By selecting $lr = 0.01$, the model benefits from faster learning while maintaining stability, setting the groundwork for fine-tuning in subsequent training stages.

4. Anchor Matching Criteria

Anchor matching criteria for cars:

- Positive anchors: IoU ≥ 0.6.
- Negative anchors: IoU < 0.45.
- Ignore region: Anchors with $0.45 \leq$ IoU < 0.6.

These thresholds ensured a balanced trade-off between precision and recall, particularly for detecting large and consistently shaped objects like cars.

5. Data Augmentation

Data augmentation was applied to enhance the model's robustness to variations in object orientation, size, and position, which are commonly encountered in real-world scenarios. The following augmentation techniques and values were used:

- **Random Rotations**: $\Delta\theta \sim \mathcal{U}[-\pi/10, +\pi/10]$, simulating natural variations in object orientation.
- **Random Translations**: $\Delta x, \Delta y, \Delta z \sim \mathcal{N}(0, 1)$, accounting for positional shifts due to sensor noise or object movement.
- **Global Scaling**: Scaling factor $s \sim \mathcal{U}[0.95, 1.05]$, reflecting changes in object size due to distance or perspective.
- **Global Rotations**: Global rotation angle $\theta_g \sim \mathcal{U}[-\pi/4, +\pi/4]$, introducing larger scene-wide rotational variations to improve generalization.

These augmentations were designed to mimic the diversity of data encountered in autonomous driving environments. By introducing controlled randomness, the model learns to be invariant to small changes in orientation, position, and scale, reducing overfitting to the training data.

6.Loss Function

As mentioned in paper[base paper] the loss function is a combination of classification and regression losses, defined as:

$$
\begin{aligned}
L = \alpha \frac{1}{N_{\text{pos}}} \sum_i L_{\text{cls}}(p_i^{\text{pos}}, 1) \\
+ \beta \frac{1}{N_{\text{neg}}} \sum_j L_{\text{cls}}(p_j^{\text{neg}}, 0) \\
+ \frac{1}{N_{\text{pos}}} \sum_i L_{\text{reg}}(u_i, u_i^*)
\end{aligned}
\tag{2}
$$

The hyperparameters are set as $\alpha = 1.5$ and $\beta = 1.0$. The value of $\alpha = 1.5$ prioritizes learning from positive anchors, which is essential for improving object detection performance by focusing on relevant regions. Meanwhile, $\beta = 1.0$ ensures balanced weighting between the regression loss and the classification loss for negative anchors, thereby preventing overfitting to negative regions and maintaining robust gradient flow during training.

C. *Training*

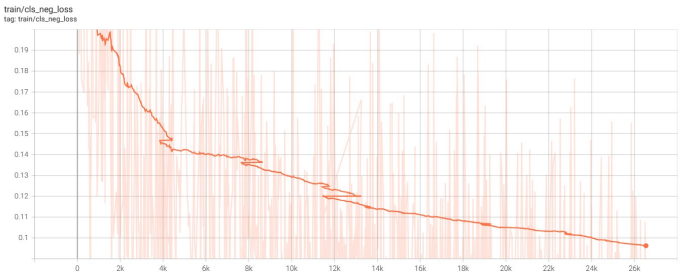

Fig. 5. Training loss for negative anchors.

VoxelNet was evaluated on the Car category from the KITTI 3D object detection benchmark, which comprises 7,481 training samples (images/point clouds) and 7,518 test samples.Since the ground truth for the test set is not available and access to the KITTI test server is limited, a common evaluation protocol was adopted. The 7,481 training samples were split into two subsets: 3,712 samples for training and 3,769 samples for validation. To ensure a fair evaluation, the split was conducted such that samples from the same driving sequence were not included in both the training and validation sets. The model was trained on the training set and evaluated on the validation set for the Car category as moderate.

The classification loss on negative anchors (L_{cls}) in the train datasets highlight the impact of the chosen hyperparameters. The consistent decline in Fig. 5 over the training epochs demonstrates that the model effectively learns to differentiate non-object regions, supported by the balanced weight $\beta = 1.0$, which ensures the classification loss for negative anchors contributes appropriately without overwhelming the regression task. The validation loss closely aligns with the validation loss, indicating strong generalization, which is further enhanced by the prioritization of positive anchors through $\alpha = 1.5$, reducing false positives and improving detection precision. Furthermore, the use of data augmentation, including random rotations, translations, and scaling, introduces diversity in

training samples, helping the model generalize to real-world variations and preventing overfitting.

V. Results

The performance of the proposed VoxelNet-based 3D object detection model was evaluated using Intersection over Union (IoU) as the primary metric. The experiments were conducted on 1,000 images from the KITTI test set. The model was trained and evaluated for both 10 and 15 epochs to analyze the impact of additional training epochs on detection accuracy.

The IoU results for 3D bounding box predictions are summarized below:

For the ”moderate” difficulty level, the model achieved an avg IoU of 0.58 after 10 epochs. While the performance was satisfactory for many images, there were instances where the predicted bounding boxes had no overlap with the ground truth. This resulted in a zero IoU for those cases, highlighting challenges in detecting objects with partial occlusion or sparse point clouds. These outlier cases contributed to a lower overall average IoU. Extending the training duration to 15 epochs improved the IoU to 0.644. The additional training allowed the model to better refine its feature learning and region proposal processes, reducing the number of cases with no overlap. This improvement indicates that longer training helps the model generalize better, even for moderately challenging scenarios.

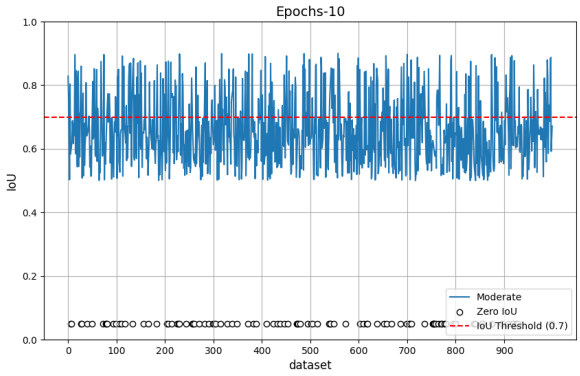

Fig. 6. IoU for test images with Epoch-10

Fig. 6 and Fig. 7 depict the IoU values for a dataset of 1000 images for trained model with 10 and 15 epochs respectively. The IoU values above 0.7, represented as the red dashed line, indicating accurate detections. IoU's between 0 and 0.45 are visualized as white markers with black edges at the bottom of the plot, signifying failed detections. The remaining IoUs lie between 0.5 and 0.7, showing predictions with partial overlaps. The distribution of IoU values highlights the variability of model performance across the dataset.

The comparison between the models trained for 10 and 15 epochs demonstrates a clear improvement with extended training. The model trained for 10 epochs achieved an average IoU of 0.58 for the ”moderate” difficulty level, but significant instances of zero IoU, due to challenges like partial occlusions

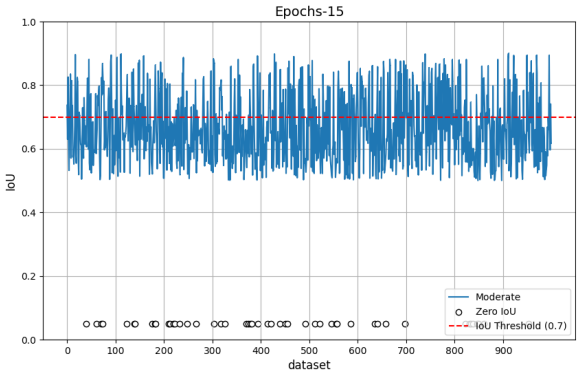

Fig. 7. IoU for test images with Epoch-15

or sparse point clouds, lowered the overall performance. In contrast, the model trained for 15 epochs showed a higher average IoU of 0.644, with fewer zero IoUs and improved consistency across predictions. The additional training allowed better feature extraction and region proposal refinement, reducing failed detections and enhancing accuracy, making the 15-epoch model more reliable for moderately challenging scenarios.

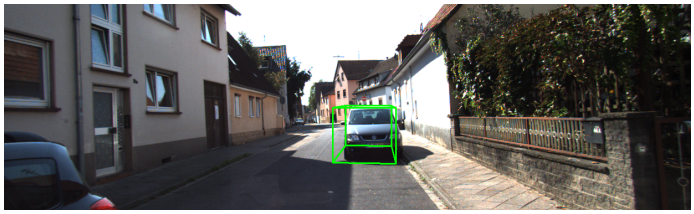

Fig. 8. 2D Image with detected 3d bounded boxes

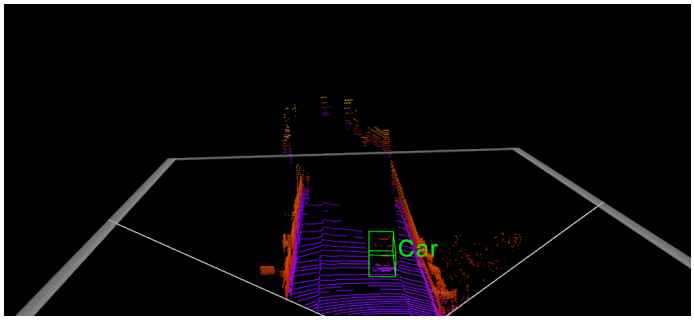

Fig. 9. Lidar with detected car object

The trained model processes these inputs to predict objects in the scene, generating 3D bounding boxes for detected objects-cars. These bounding boxes are then visualized in two modalities for evaluation. In the image, the predicted 3D bounding boxes are projected into the 2D image plane, outlining the detected objects for visual confirmation in the 2D domain. In the LiDAR point cloud, the 3D bounding boxes are overlaid onto the point cloud data, encapsulating the

Fig. 10. 2D Image with detected 3d bounded boxes

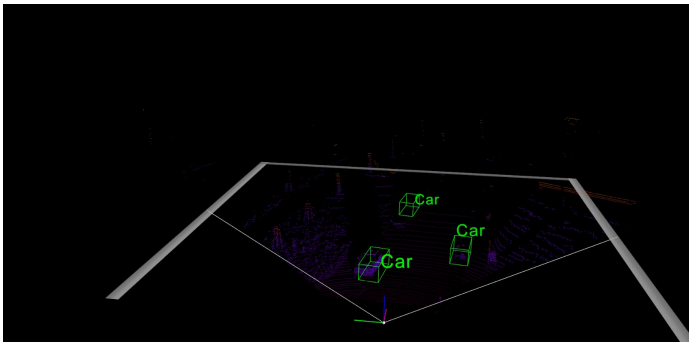

Fig. 11. Lidar with detected car object

detected objects in 3D space to assess spatial accuracy. This dual-modality visualization ensures consistency and validates the model's predictions in both the image and LiDAR data, highlighting its capability to accurately detect and localize objects in real-world scenes.

While extending training from 10 epochs to 15 epochs led to an improvement in the average IoU from 0.58 to 0.644 for the "moderate" difficulty level, there are still cases where predictions fall short, particularly in detecting partially occluded objects or handling sparse point clouds in Fig. 10. The incremental improvement observed between 10 and 15 epochs suggests that additional training could further refine the model's feature extraction and localization capabilities, leading to more consistent and accurate predictions across diverse scenarios.

VI. Limitations

- Even thought the model implemented in this project was performed on a RTX 3060 6GB of GDDR6 of VRAM, training with around 7.5k 3D point-cloud sample data caused a significantly high computation cost/ strain during the model training.
- Originally, the VoxelNet network were trained for about 160 epochs, which required huge computation power and extensive duration. Considering the limited computation power available to train the model under the given time, we were only able to train the model with 10, 15, and 20 epochs to derive appreciable performance.
- As mentioned earlier, training with entire KITTI dataset of about 7.5k samples was quite demanding and laborious, therefore we had to train the model by parts considering about 3.8k data samples at a time.
- Despite overcoming all the above stated limitations, the overall time taken or consumed to train an entire model with 3.8k sample data and testing them using another 3.8k data points including validation took about 11.8 intermittent hours (approx.) provided that the computer didn't crash or return memory overflow during the training.

VII. Conclusion

The key result of the study is that the model trained for 15 epochs achieved an average IoU of 0.644 on the "moderate" difficulty level, compared to 0.58 for the model trained for 10 epochs. Adding the RoI layer to VoxelNet improved the precision of object detection by refining the features from RPN proposals, resulting in better localization. It required careful analysis of the RPN outputs and smoothly merging them with the RoI layer. This included creating a fixed-size output from the RoI Pooling operation that could connect properly with the fully connected layers for classification and regression. This step was key to improving the accuracy of object detection while keeping the system efficient. Setting up the environment for implementation was also challenging. Tasks like configuring CUDA, installing the correct drivers, and ensuring compatibility with deep learning frameworks like tensorflow were essential but time-consuming. Solving these issues provided a better understanding of the tools and setup needed for large-scale experiments.

There are many ways to improve this work in the future. Adding more refinement stages could make detection better, especially for small or far-away objects. Replacing the dense 3D convolutions in VoxelNet with sparse convolutions could reduce computation time and make the system faster. The ideas from combining RoI Pooling with voxel-based networks can also be used in other areas like robotics, underwater vehicles, or drones, where understanding 3D space is important. Making the system faster could also open doors for real-time applications like self-driving cars and augmented reality. Overall, this project was a great learning experience in advanced 3D object detection and showed how to effectively add new features to existing systems.

VIII. Contributions

The contributions to the project were divided based on the key components of the VoxelNet-based 3D object detection framework:

- **Manoj Kumar Selvaraj**: Primarily worked on the *Voxel Feature Encoding (VFE) layer*, focusing on feature extraction from raw point cloud data and encoding it into voxel-wise representations.
- **Anbarasan Kandasamy**: Contributed to the implementation of the *Convolutional Neural Network (CNN) middle layers* and the *Region Proposal Network (RPN)*, which aggregated voxel-wise features and generated 3D region proposals.
- **Hamsaavarthan Ravichandar**: Worked on the *Region of Interest (RoI) pooling layer* and the *evaluation metrics*,

ensuring accurate refinement of the 3D bounding box predictions and thorough performance analysis.

REFERENCES

[1] Alireza Ghasemieh, Rasha Kashef, 3D object detection for autonomous driving: Methods, models, sensors, data, and challenges, Transportation Engineering, Volume 8, 2022.

[2] Rayan Potter, Applications and Challenges with 3D Point Cloud Data for LIDARs, ANOLYTICS, 2021.

[3] Yin Zhou, Oncel Tuzel, VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection, arXiv:1711.06396v1 [cs.CV], Nov, 2017.

[4] Adam Cuellar, Article: VoxelNet End-To-End Learning for Point Cloud Based 3D Object Detection, Machine Intelligence and Deep Learning, May, 2022.

[5] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In IEEE CVPR, 2017.

[6] B. Li. 3d fully convolutional network for vehicle detection in point cloud. In IROS, 2017.

[7] B. Li, T. Zhang, and T. Xia. Vehicle detection from 3d lidar using fully convolutional network. In Robotics: Science and Systems, 2016.

[8] Changyu Zeng, Wei Wang, Anh Nguyen, Jimin Xiao, Yutao Yue, Self-supervised learning for point cloud data: A survey, Expert Systems with Applications, Volume 237, Part B, 2024.

[9] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 2017.

[10] Jiajun Deng, Shaoshuai Shi, Peiwei Li1, Wengang Zhou1, Yanyong Zhang, Houqiang Li, Voxel R-CNN: Towards High Performance Voxel-based 3D Object Detection, arXiv:2012.15712v2 [cs.CV], 5 Feb, 2021.

[11] Zhou Y, Tuzel, Voxelnet: End-to-end learning for point cloud based 3d object detection. In CVPR, 4490–4499, 2018.

[12] Qi C. R; Su H, Mo K, Guibas L J, Pointnet: Deep learning on point sets for 3d classification and segmentation. In CVPR, 652–660, 2017.

[13] Qi C. R; Su H, Mo K, Guibas L J, Point-net++: Deep hierarchical feature learning on point sets in a metric space. In NeurIPS, 5099–5108, 2017.

[14] Yang Z, Sun Y, Liu S, Shen X, Jia J, STD: Sparse-to-dense 3D object detector for point cloud. In ICCV, 1951–1960, 2019.

[15] M. Enzweiler and D. M. Gavrila. A multilevel mixture-of-experts framework for pedestrian classification. IEEE Transactions on Image Processing, 20(10):2967–2979, Oct 2011.

[16] A. Gonzlez, D. Vzquez, A. M. Lpez, and J. Amores. On-board object detection: Multicue, multimodal, and multiview random forest of local experts. IEEE Transactions on Cybernetics, 47(11):3980–3990, Nov 2017.

[17] Girshick R, Fast r-cnn. In ICCV, 1440–1448, 2015.

[18] Git Repository: https://github.com/qianguih/voxelnet