2023

A Study of Attention-Free and Attentional Methods for LiDAR and 4D Radar Object Detection in Self-Driving Applications

King Wah Gabriel Chan

A study of Attention Free and Attentional methods of LiDAR and 4D radar object detection for self driving

CONNECTICUT
COLLEGE

Gabriel Chan
Department of Computer Science
Connecticut College

Supervisor
Ozgur Izmirli PhD

In partial fulfillment of the requirements for the degree of
Bachelor of Arts in Computer Science

May 2023

Acknowledgements

I'm extremely grateful to Professor Izmirli for being my advisor this year for providing valuable ideas and feedback to my journey throughout this thesis. Many thanks to Professor Chung and Professor Lee for being my readers and providing editing help even with their limited time.

I would also like to thank my previous collaborators Jianning Deng and Ivan Zhong at the MAPS lab in the University of Edinburgh for their invaluable support and guidance as I entered this field. Their expertise and willingness to collaborate greatly contributed to my growth as a researcher and the knowledge on this topic.

Abstract

In this thesis, we re-examine the problem of 3D object detection in the context of self driving cars with the first publicly released View of Delft (VoD) dataset [1] containing 4D radar sensor data. 4D radar is a novel sensor that provides velocity and Radar Cross Section (RCS) information in addition to position for its point cloud. State of the art architectures such as 3DETR [2] and IASSD [3] were used as a baseline. Several attention-free methods, like point cloud concatenation, feature propagation and feature fusion with MLP, as well as attentional methods utilizing cross attention, were tested to determine how we can best combine LiDAR and radar to develop a multimodal detection architecture that outperforms the baseline architectures trained only on either modality alone. Our findings indicate that while attention-free methods did not consistently surpass the baseline performance across all classes, they did lead to notable performance gains for specific classes. Furthermore, we found that attentional methods faced challenges due to the sparsity of radar point clouds and duplicated features, which limited the efficacy of the cross-attention mechanism. These findings highlight potential avenues for future research to refine and improve upon attentional methods in the context of 3D object detection.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

The recent surge in popularity of self-driving cars has sparked significant interest in the development of advanced perception systems that enable these vehicles to safely navigate complex environments. The task of autonomous driving poses a substantial challenge due to the myriad of dynamic factors involved, such as diverse road conditions, unpredictable behavior of other road users, and ever-changing environmental factors. Ensuring the safety of passengers, pedestrians, and other road users is of paramount importance, as any failures or inaccuracies in the detection and tracking system can lead to severe consequences.

Sensors are the cornerstone of self-driving cars' detection systems, as they allow the vehicle to detect, locate, and interpret its surroundings. A wide range of sensors, including cameras, LiDAR (Light Detection and Ranging), radar, ultrasonic, and GPS, are employed to capture a comprehensive understanding of the environment. Among these, LiDAR and cameras have typically been the main sensors used for perception. However, 4D radar sensors have emerged as particularly promising for 3D object detection due to their ability to provide precise distance, velocity, and object shape information.

To achieve a high level of safety and efficiency in autonomous driving, it is crucial to develop a robust and accurate object detection model that effectively utilizes the information provided by these sensors. This requires an object detection model that optimally fuses LiDAR and radar sensor data, which is the primary focus of our research. By exploring various methods to integrate the complementary strengths of these two sensor modalities, we aim to devise a multimodal object detection model that surpasses the performance of current state-of-the-art approaches, thereby contributing to the advancement of self-driving car detection and tracking systems and, ultimately, their widespread adoption.

1.2 Context of the Study

The idea of self-driving vehicles dates back to the early days of computer science, with concepts and prototypes appearing in the 1920s and 1930s [16]. However, it wasn't until the 1980s that significant progress was made in the development of self-driving technology, and it wasn't until the 2000s that the field truly began to take off. Here is a brief history of self-driving from a computer science perspective:

1980s - Early research: In the 1980s, researchers began to explore the idea of self-driving vehicles. One of the first notable projects was the Navlab project at Carnegie Mellon University [17], which developed a self-driving van that used computer vision and neural networks to navigate.

2000s - DARPA Grand Challenge: The DARPA Grand Challenge, first held in 2004 [18], was a pivotal moment in the history of self-driving vehicles. The challenge offered a significant cash prize to the team that could develop an autonomous vehicle capable of driving through a designated course. The first year of the competition saw no successful finishes, but subsequent years saw significant progress, with a team from Stanford University winning the 2005 challenge [19] and a team from Carnegie Mellon winning the 2007 challenge [6].

2010s - Commercialization: In the 2010s, self-driving technology began to be commercialized, with companies like Waymo, Uber [20], and Tesla [21] investing heavily in the technology. In 2018, Waymo (a subsidiary of Google) launched a self-driving taxi service in Phoenix, Arizona [22], marking the first time self-driving vehicles were available to the public.

2020s - Continued development: Today, self-driving technology continues to evolve, with advances being made in areas such as machine learning, computer vision, and sensor technology. While there are still many challenges to be overcome, including regulatory issues and the need for further technological development. One aspect is object detection. Object detection is the process of identifying and localizing objects of interest in an image or a video stream, and is a critical component of many computer vision systems. In the context of self-driving cars, object detection is particularly challenging due to the need to identify a wide range of objects in a variety of environments in real time.

The importance of object detection in self-driving cars can be understood in the context of the complex and dynamic nature of the driving environment. Self-driving cars must be able to accurately detect and track a wide variety of objects, including other vehicles, pedestrians, bicyclists, road signs, and traffic signals, among others. Attempts to capture the environment often include the use of a wide range of sensors like cameras, LiDAR, radar (radio detection and ranging), and ultrasonic, and other sensors [4].

RGB cameras and LiDAR are two most common sensor technologies used in self-driving cars for perception. RGB cameras capture images of the surrounding environment using visible light, while LiDAR emits laser beams in a rotating pattern, scanning the surrounding environment to detect objects. As the laser beams hit objects in the environment, they bounce back to the sensor and are recorded as points in 3D space [23]. The LiDAR sensor then uses algorithms to process these points and generate a 3D point cloud map of the environment. The point cloud map is essentially a collection of 3D points, each representing the location of an object in the environment. Each point has an x, y, and z coordinate, representing its position in 3D space, as well as an intensity value representing the return strength of the beam.

One of the main advantages of RGB cameras is their low cost and wide availability, making them an attractive sensor option for mass-market adoption. In addition, RGB cameras can capture color and texture information about objects, which can be useful for identifying objects that are difficult to detect with LiDAR alone, such as road signs and traffic lights. While an RGB camera may be able to capture the textual information of a sign (e.g. speed limit), LiDAR can only capture the shape of the sign. Hence why RGB cameras can be a cost effect way to provide a rich source of visual data for object detection algorithms.

However, there are also some drawbacks to using RGB cameras in self-driving cars. One of the main challenges is their sensitivity to lighting conditions, which can affect their performance and accuracy. For example, changes in lighting due to weather conditions or the position of the sun can make it difficult for cameras to accurately detect objects. In addition, since RGB cameras essentially project light onto a 2D plane to form an image, very little 3D information can be inferred natively from a single image. While there are ways to mitigate this by using multiple cameras [24], RGB cameras fundamentally have a limited depth perception compared to LiDAR, which can make it difficult to accurately estimate the distance and speed of objects. Finally, the quality of the image captured by RGB cameras can be affected by factors such as motion blur and focus, which can impact their usefulness for object detection.

LiDAR, on the other hand, provides accurate depth information of the environment with a high definition point cloud with hundreds of thousands of points. This provides valuable information about the shape, size and orientation of objects in a scene, which can be used to identify and classify objects. In addition, LiDAR can operate in a wide range of lighting and weather conditions unlike RGB cameras, making it a reliable sensor for self-driving cars.

However, there are also some drawbacks to using LiDAR in self-driving cars. One of the main challenges is the cost of the sensor, which can be prohibitively expensive for mass-market adoption. For example, the flagship LiDAR sensor from Velodyne can cost up to $75,000, while even the cheapest alternatives by companies like Luminar Technologies still cost around $1,000 [25]. Another drawback is the size and weight of the sensor, which can limit its placement and integration into the vehicle. Finally, LiDAR provides little information regarding the speed of objects in the environment, as the intensity value of the point is the intensity of the reflected laser beam and is largely dependent on the distance. As a result, it is difficult to determine the type of material and the velocity of an object in a LiDAR point cloud.

1.2.1 4D Radar Sensor

Unlike traditional radar sensors, which have very low spatial resolution and are therefore able to capture data only in the ground plane in the environment, as illustrated in Figure 1.1, 4D radar captures additional information about the height, velocity, and radar cross-section (RCS) of objects. This additional information has the potential to significantly improve the accuracy of object detection and tracking, making 4D radar an exciting emerging sensor for autonomous driving applications.

4D radar sensors work by emitting electromagnetic waves, which bounce off

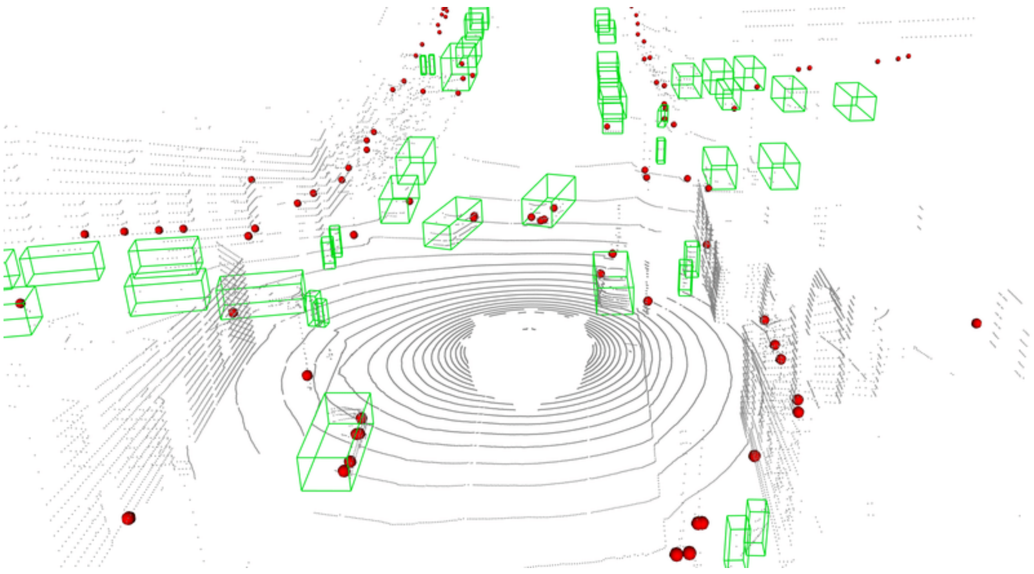


Figure 1.1: A frame from the dataset Nuscenes [4], LiDAR points are displayed in grey and radar points in red, 3D ground truth boxes are in green. Image taken from [5]

of objects in the environment and are then detected by the sensor. The time it takes for the waves to bounce back to the sensor provides information about the distance to the object, while the Doppler effect provides information about the object's velocity. The radar cross-section, or RCS, of an object refers to the amount of electromagnetic energy that is reflected back to the sensor, which can be used to infer the size and shape of the object.

Like LiDAR sensors, 4D radar data is most often captured in a point cloud format, and is similarly robust to adverse weather conditions and poor lighting conditions. However, while they have a much higher resolution (higher number of points) than traditional radar sensors, a single scan from a 4D radar sensor still only contains on the order of hundreds of points. While it is able to provide data like velocity and RCS that LiDAR and RGB cameras cannot, the sparsity of the point cloud prevents 4D radar sensors from being used independently for object detection purposes, especially for safety critical tasks like self driving. Therefore one of the main objects is to investigate how we could integrate radar with other sensors to enhance object detection capabilties.

1.2.2 Classical approaches to LiDAR object detection

While LiDAR has always played a part in attempts to solve self driving, the algorithms employed to use the acquired point cloud data has evolved over time. In team Carnegie Mellon University's vehicle in the 2007 DARPA Urban Challenge[6], they employed various non deep learning methods such as Haar wavelet transforms, heuristic edge detection with adaptive thresholding, and dynamic programming methods for optimal line splitting to detect road edges. Figure 1.2 illustrates sample output using the Haar wavelet transform method.

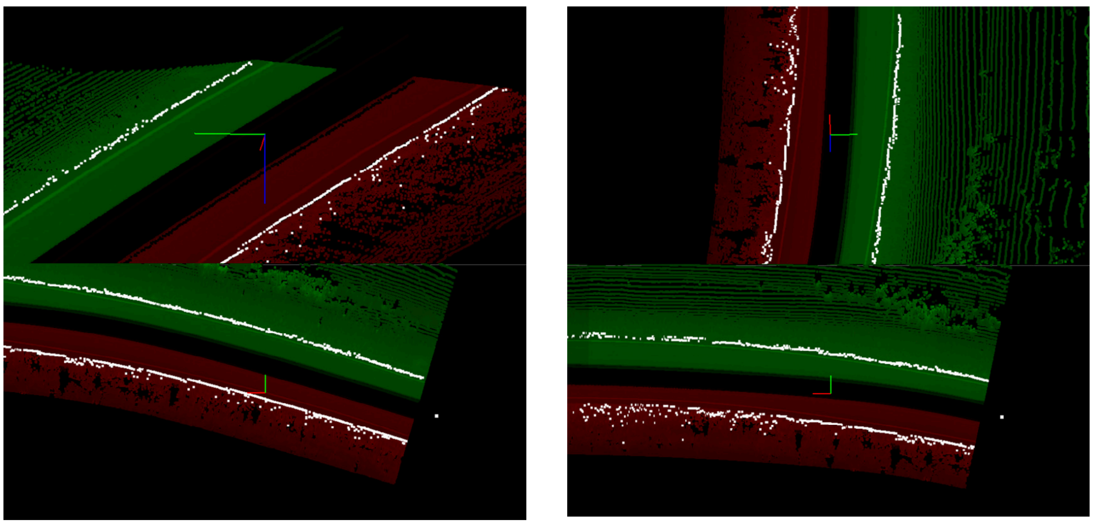


Figure 1.2: Example output of the Haar road edge detection algorithm operation on point cloud data. Image taken from [6]

Another method used for object detection utilized the elevation differences of points in a scan coupled with a cost function to determine the traversability of the terrain to build an obstacle map of its environment, as illustrated in Figure 1.3. As the authors stated themselves, this approach had many limitations as well.

For example, the algorithm considers foliage as obstacles and does not explicitly locate the object in the point cloud.

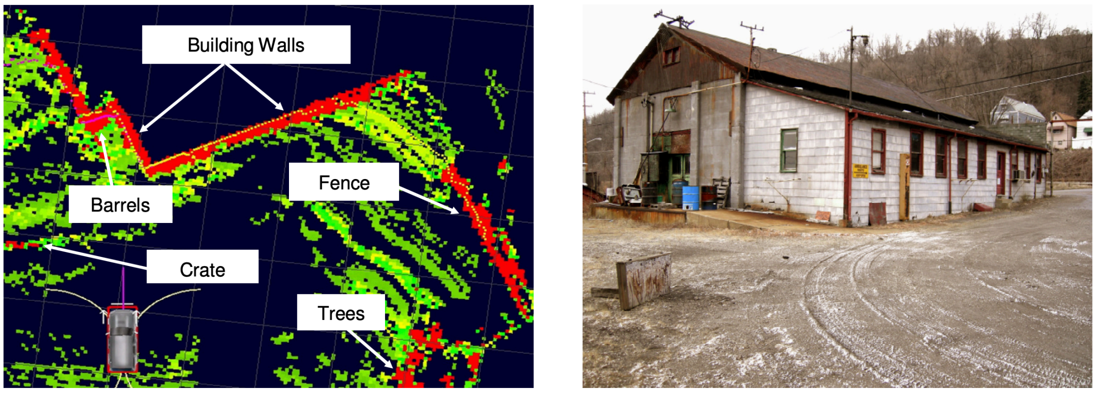


Figure 1.3: Obstacle map constructed with a LiDAR scan. Image taken from [6]

Algorithms were deployed to detect objects for each of the sensors, and while this may have increased the detection robustness, their approach to detecting vehicles using LiDAR in isolation was rather simple. It utilizes an algorithm to cluster LiDAR points into linear segments and convex 90 degree corner objects, as shown in Figure 1.4. While this algorithm may be sufficient to detect basic shapes, it would have been difficult to classify the different detected objects for further downstream tasks. Moreover, the detected object is only considered from the Bird's Eye View (BEV) perspective, neglecting the vertical dimension.

1.2.3 Modern Approaches: Deep Learning

With advances in LiDAR sensors, deep learning architectures and hardware, complex tasks such as object detection on point clouds has largely shifted towards using deep learning models. The shift accelerated when PointNet[26] and PointNet++[27] were published. They were two seminal papers in deep learning architectures for processing point clouds, which have had a significant impact on the field of 3D perception.

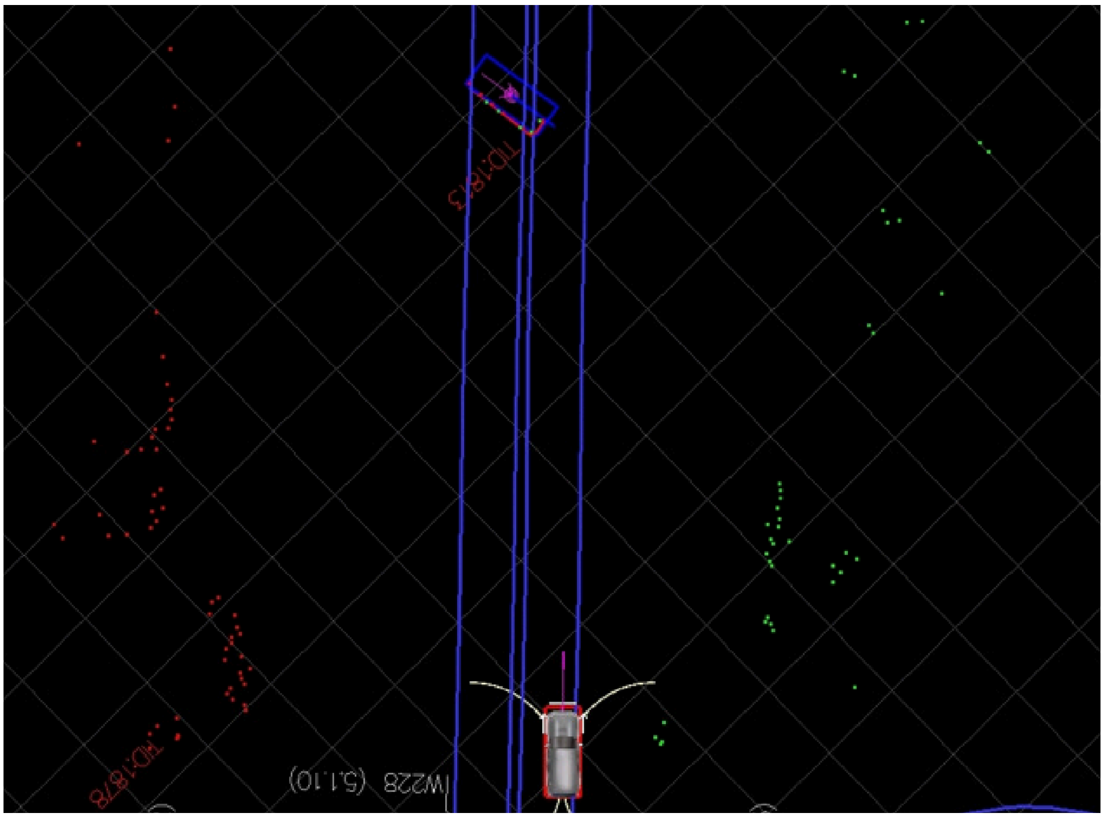


Figure 1.4: Detected vehicle is outlined in the blue rectangle, and the red line are the line segments formed by the algorithm. Image taken from [6]

Prior to the introduction of PointNet[26] in 2017, point clouds were typically processed using hand-crafted features, such as point feature histograms [28]. These techniques were often computationally expensive and required domain-specific knowledge, making them difficult to generalize to different applications.

Furthermore, processing point clouds come with their own challenges, the primary one being that points can be ordered arbitrarily in a point cloud as they are essentially stored as 2D arrays. E.g. a point cloud consisting of 5 points and their 3D coordinates would be a 5×3 array. Thus any model must be invariant to all $n!$ permutations of a point cloud of size n. Additionally, it is not guaranteed that each LiDAR scan will contain the same number of points. Contrasting to images where the image resolution is typically predefined and can be manipulated.

PointNet was introduced to address these issues, by introducing a neural network architecture that can directly process raw point cloud data without the need for hand-crafted features. The architecture uses a combination of multi-layer perceptrons (MLPs) and max pooling to learn a global feature representation for the point cloud. This representation can be used for tasks such as classification, segmentation, and object detection. PointNet achieved state-of-the-art (SOTA) performance on several benchmark datasets, and is now considered a foundational method for point cloud processing. The PointNet model can be summarized as follows. Given an unordered point set $\{x_1, x_2, \ldots, x_n\}$ with $x_i \in \mathbb{R}^k$, one can define a set function $f : \mathcal{X} \to \mathbb{R}$, that maps a set of points to a vector:

$$f(x_1, x_2, \ldots, x_n) = \gamma \left(\underset{i=1,\ldots,n}{\mathrm{Max}} \{h(x_i)\} \right) \tag{1.1}$$

where γ and h are typically MLPs, and $\mathcal{X} = (M, d)$ is a discrete metric space, with $M \subseteq \mathbb{R}^k$ being the set of points and d being the Euclidean metric. PointNet was the first published deep learning model capable of approximating any continuous set function.

PointNet++ [27] was introduced in 2017 as an extension of the PointNet architecture, which addresses the issue of scale invariance. Its architecture is displayed in Figure 1.5. While PointNet can process point clouds of any size, it is not able to capture fine-grained details at different scales. PointNet++ addresses this issue by using a hierarchical neural network architecture to process point clouds at different scales. The network uses a set abstraction module to group points into local regions. For a point cloud with N points, M points are chosen with $M < N$ via a downsampling algorithm like farthest point sampling (FPS), and for each M points, group k nearest points within some sampling radius r, and pass those k points and their features through a PointNet to acquire a point cloud with M points and new features. This process constitutes one set abstraction layer, and this is typically repeated multiple times. Furtheremore, one technique that is often used in practice is called multiscale grouping (MSG). This is modification where the sampling procedure is repeated twice within a set abstraction layer, with sampling radii r_1 and r_2 and sampled points k_1 and k_2 respectively. The features at each scale are passed through a PointNet respectively before concatentation and passed through MLP once more. Pointnet++ and the set abstraction module has continued to be key design in state of the art object detection models that followed its publication.

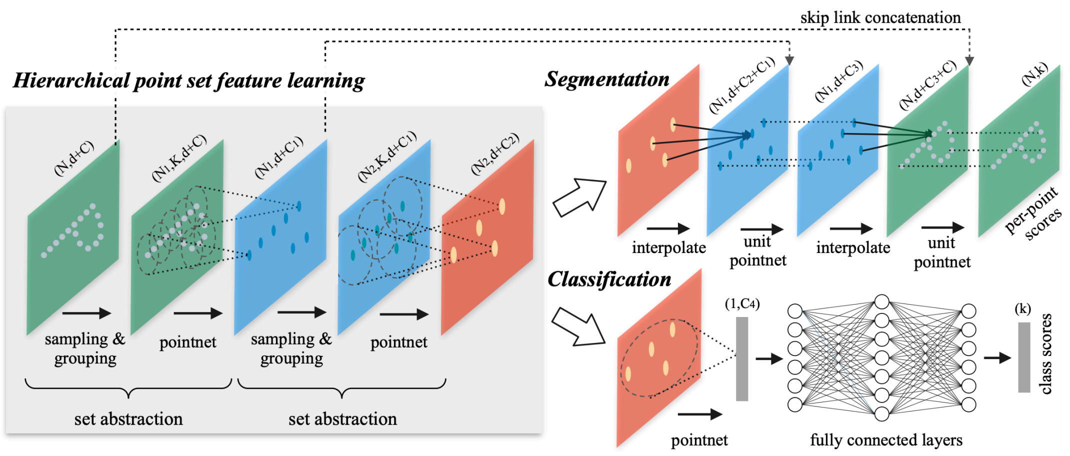


Figure 1.5: Pointnet++ Architecture. Left side illustrates repeated applications of the set abstraction layer, right side illustrates how the resulting point features can be used in segmentation and classification tasks.

1.2.4 Attention and Transformers

A Transformer is a deep neural network architecture that was introduced in the paper ”Attention Is All You Need” by Vaswani et al. in 2017 [29]. It was originally designed specifically for natural language processing (NLP) tasks such as machine translation, language modelling, and text generation, but has since seen usage across different fields like computer vision, becoming a foundational model within the field of deep learning.

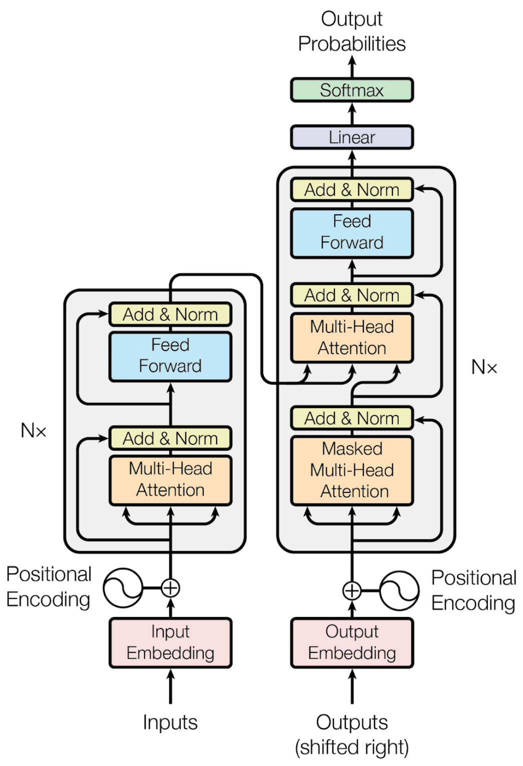


Figure 1.6: Transformer architecture introduced by Vaswani et al.

At the heart of the transformer is the attention mechanism, which allows the model to focus on the most relevant parts of the input sequence when making predictions. In a traditional recurrent neural network (RNN), the model processes the input sequence one element at a time, using a hidden state to summarize the information learned so far. However, this approach has several drawbacks, including difficulty in capturing long-term dependencies and slow training times.

The attention mechanism in the transformer addresses these issues by allowing the model to directly attend to all elements of the input sequence at once, without the need for recurrent connections. In the transformer model, the architecture of which is illustrated in Figure 1.6, each layer consists of two sub-layers: a multi-head self-attention mechanism and a feed-forward neural network. The self-attention mechanism allows the model to attend to different parts of the input sequence with varying degrees of importance, while the feed-forward network applies a non-linear transformation to the attention output. Given a set of query vectors $Q \in \mathbb{R}^{n \times d_k}$, a set of key vectors $K \in \mathbb{R}^{n \times d_k}$, and a set of value vectors $V \in \mathbb{R}^{n \times d_v}$, the scaled dot product attention computes the weighted sum of value vectors as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where n is the number of elements in the input sequence, d_k and d_v are the dimensions of the key and value vectors respectively, and softmax is the softmax function that normalizes the attention scores across the input sequence. The dot product QK^T is scaled by $\sqrt{d_k}$ to ensure that the magnitude of the dot product is not affected by the dimensionality of the input sequence.

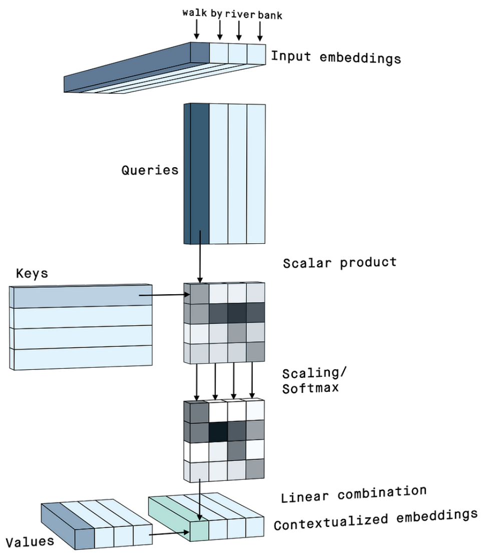


Figure 1.7: Illustration of the attention mechanism. Image taken from [7]

The special case where $Q = K = V$ is most frequently called self attention, as the mechanism re-weights the input feature vectors using the input vectors themselves. This mechanism enabled back propagation to differentially determine which features are important and updating the vectors themselves accordingly. This process is illustrated in figure 1.7.

In practice however, multi-headed self attention is more frequently used, where each feature vector $x \in \mathbb{R}_k^d$ is linearly projected into n different vectors with dimension $\frac{d_k}{n}$, where the n final vectors are concatenated back together, this is illustrated in Figure 1.8. The authors point out that this enabled the network to learn multiple relationships between the set of vectors.

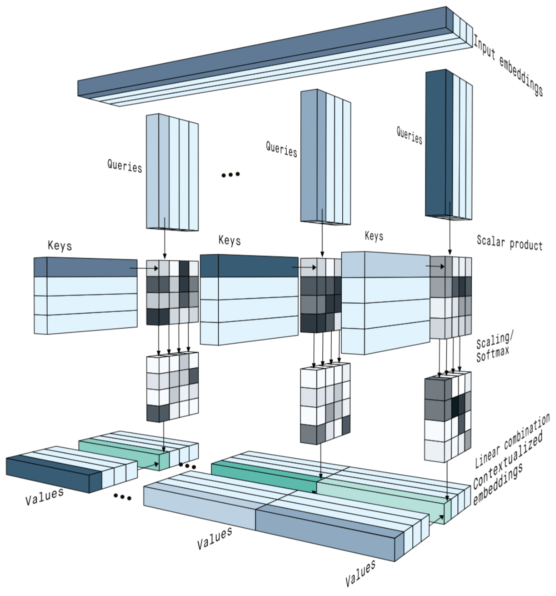


Figure 1.8: Illustration of the multiheaded attention mechanism. Image taken from [7]

Notably, the self-attention mechanism is naturally permutation invariant because it uses dot product and softmax operations to compute the weights between every pair of input vectors, without considering their position in the set. This means that each feature vector is compared to every other input vector in the set, and the resulting weights are used to compute a weighted sum that captures the relationships between all vectors, regardless of their order.

This permutation invariance is a desirable property for some tasks, where the order of the input features are not important. It allows the self-attention mechanism to capture the relationships between all pairs of input vectors, regardless of their position.

In many natural language processing tasks however, such as language modeling and machine translation, the order of the tokens is critical to the meaning of the text. To address this issue, additional vectors called positional embeddings are added to the input vectors to encode their position in the sequence. These positional embeddings are learned during the training process and are added to the input embeddings before they are fed into the self-attention layer.

The positional embeddings provide a way for the model to distinguish between tokens based on their position in the sequence. By adding the positional embeddings to the input embeddings, the model can differentiate between tokens with the same embedding but different positions in the sequence.

1.3 Objectives and Contributions

The objective of this thesis is therefore to explore the following three main questions:

1. Is it possible to utilize a vanilla transformer-like model to not only perform object detection on outdoor dataset, but also incorporate additional information from an extra modality?
    - The motivation being the simplicity of the transformer model and the success of modern architectures based on the transformer. This question is most tempting of all as it would further illustrate the flexibility of the transformer architecture to multimodal point cloud problems.

2. If we cannot achieve (1), how can we then take a state of the art architecture for LiDAR based object detection and improve it such that it can process both LiDAR and radar information without the use of the attention mechanism?
    - Relative to developing an entirely new architecture, it would be advantageous to build off state of the art methods and incorporate techniques such as transfer learning to tackle this problem.
3. If we use the attention mechanism in the form of cross attention, is it possible to outperform methods that do not use attention?
    - Given the supposed superiority of the attention mechanism, can we incorporate into our methods building off the state of the art architectures to further improve performance?

and we summarize our contributions in exploring these questions as follows:

1. Vanilla Transformer-like models like [2] cannot achieve reasonable performance for outdoor datasets due to the following reasons. Firstly, randomly selecting points as seed points as the decoder input, which ultimately becomes the prediction proposals, is poor strategy for outdoor datasets due to how little points an object may have. This is compared to indoor datasets where point clouds are much more smaller and therefore more dense. Furthermore, even when only using the encoder coupled with downsampling, the resulting proposals are of much more lower quality compared to architectures based on Pointnet.
2. We introduced three attention free methods: (a) point cloud concatenation, (b) feature interpolation, and (c) feature fusion with MLP. These methods are advantageous given the little additional computations requires. This

because the operations introduced are either performed on the raw input point cloud for (a) and (b), or utilize a single MLP at the final stage. While (c) was able to improve the overall average performance above the baseline method, much like (a) and (b) it suffered from a loss of performance in the car class.

3. We introduce the Multimodal Set Abstraction (MMSA) layer which integrates variations of cross attention, from a single layer, to stacked transformer encoder with normalization layers and MLPs. We find that the best variation performs similar to the feature fusion with MLP. Investigations attribute this phenomenon to the fact that nearby radar proposal points in the final layer frequently have the same features. As a result, when applying cross attention with LiDAR to these duplicated features, the net result is essentially applying the identity matrix to the set of radar features.

Chapter 2

State of the Art

2.1 Notable 3D object detection architectures

In the following section, we review a selection of relevant architectures developed in recent years. The model described in section 2.1.1 was the first work that utilized PointNet as the base model, VoteNet (Section 2.1.2) introduced a shifting operations that was adopted in many subsequent models like [10] [3]. PointRCNN (Section 2.1.3) is a two stage anchor box model that illustrates a common but computationally intensive approach used by earlier models similar to 2D anchor box based models. 3DSSD (Section 2.1.4) was the first model to drastically simplify the two-stage anchor based model into a single stage detector that directly regressed bounding boxes from the final point features. IASSD (Section 2.1.5) is a subsequent model that further improved the performance and efficiency of single stage detectors by introducing a novel downsampling strategy. Section 2.2 overviews several methods that adapted the vanilla transformer model for point clouds.

2.1.1 Frustum PointNet

Frustum PointNet [30] was proposed by the creator of PointNet++[27] Charles R. Qi et al. in their 2018 paper titled "Frustum PointNets for 3D Object Detection from RGB-D Data". The Frustum PointNet model builds upon the PointNet++ architecture by incorporating a frustum-based approach to object detection, which involves generating 2D bounding boxes from RGB-D data and projecting them into 3D space to create frustums. Figure 2.1 illustrates this approach. The algorithm then uses PointNet++'s set abstraction module to extract features from the point clouds within the frustums and classify the objects within them.

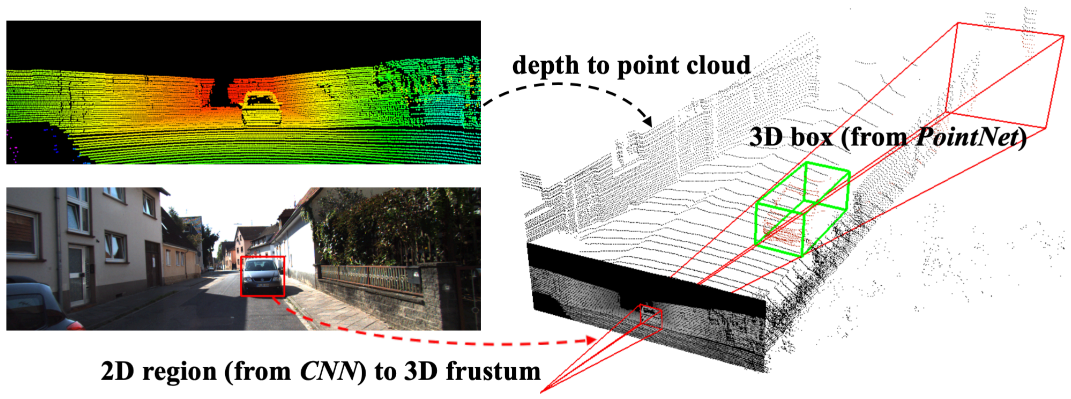


Figure 2.1: Frustum PointNet detection pipeline. 2D object region proposals are generated and extruded to a frustum. PointNets are then used onto the pointcloud within the frustum for prediction

Frustum PointNet was trained on the KITTI dataset and achieved SOTA performance, outperforming previous methods. Frustum PointNet's use of PointNet++'s set abstraction (SA) module was significant because it demonstrated the effectiveness of a SA module in handling point clouds for object detection tasks. The set abstraction module allowed Frustum PointNet to efficiently capture local and global features from point clouds, and provided a scalable framework for handling point clouds of varying sizes and densities. Since then, the set abstraction module has become a popular component in many lidar point cloud object detection algorithms, including PointRCNN, PV-RCNN, 3DSSD, and many others.

2.1.2 VoteNet

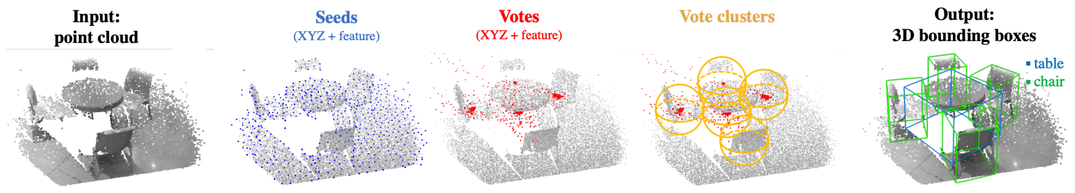


Figure 2.2: The VoteNet architecture for 3D object detection processes an input point cloud using a PointNet++ backbone network to generate a subset of seed points with extended features. These seeds create votes, which are then clustered and refined into final 3D bounding box proposals [8].

After PointNet++, many architectures like Frustum PointNet was published, utilizing the set abstraction layers as the key building block of the models. However, all such models still face an intrinsic problem of predicting a 3D bounding box from point clouds. The center of a 3D object can be far from any surface point, thus making it hard to regress accurately in one step.

The key innovation in VoteNet is the vote layer, inspired by the Hough transform [31], which is a feature extraction technique that transforms an image from the spatial domain to a parameter space. In this parameter space, the object's features are represented as points, and the object's location is identified by detecting the parameters that cause these points to align.

The Hough transform works by accumulating evidence for each parameter hypothesis from the input data, and object detection is performed by identifying the parameters with the highest evidence. In the case of Hough voting, the evidence is obtained by casting votes from individual data points, which can be image pixels or point cloud points, to the parameter space. The votes are accumulated in a voting space, and the peaks in this space correspond to the most likely object locations or poses.

VoteNet adapts the concept of Hough voting for deep learning-based 3D object detection in point clouds. Instead of relying on hand-crafted features and a predefined parameter space, VoteNet learns to generate votes for object centers directly from the input point cloud using a deep neural network. Each point in the point cloud casts a vote to predict the center of the object it belongs to, and these votes are aggregated to generate a set of potential object centers. This method is illustrated in Figure 2.2

Thus, the vote layer can then be added after several set abstraction layers to further shift points closer towards potential object centers before predicting the final bounding boxes.

2.1.3 PointRCNN

Unlike Frustum PointNet, PointRCNN [9] is a lidar-based 3D object detection algorithm that was proposed by Shi et al. in their 2019 paper titled “PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud”. It is a two-stage detection framework that aims to generate accurate object proposals and classify them into different object categories. Notably, it does not use any images during training and inference, while outperforming previous methods.

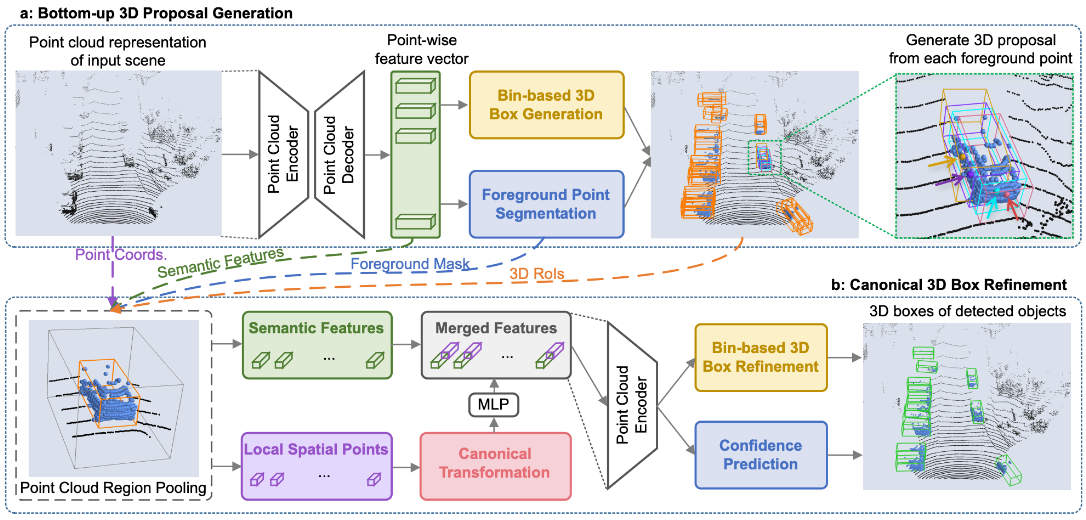

a: Bottom-up 3D Proposal Generation
Point cloud representation of input scene
Point Cloud Encoder
Point Cloud Decoder
Point-wise feature vector
Bin-based 3D Box Generation
Foreground Point Segmentation
Generate 3D proposal from each foreground point
Point Coords.
Semantic Features
Foreground Mask
3D RoIs
b: Canonical 3D Box Refinement
Point Cloud Region Pooling
Semantic Features
Merged Features
MLP
Local Spatial Points
Canonical Transformation
Point Cloud Encoder
Bin-based 3D Box Refinement
Confidence Prediction
3D boxes of detected objects

Figure 2.3: PointRCNN's two stage architecture. (a) generates 3D bounding box proposals from the raw point cloud. (b) refines the proposed bounding boxes. Image taken from [9].

In the first stage of PointRCNN, as illustrated in Figure 2.3, a region proposal network (RPN) is used to generate a set of 3D object proposals from the input point cloud. The RPN is designed to first segment the point cloud into foreground (points within an object's bounding box) and background (points not within a bounding box) points. This is done using PointNet Set Abstraction and Feature Propagation modules. Similar to the image based RCNN, it then generates 3D bounding boxes for each of these predicted points.

In the second stage of PointRCNN, as illustrated in Figure 2.3, the point-wise features and foreground mask is used to refine each of the proposed bounding boxes utilizing a PointNet style encoder. All point features inside a proposed bounding box is fed into a PointNet style encoder to refine the bounding box location and dimensions and predict a final confidence value for that bounding box.

2.1.4 3DSSD

At the time of publication, PointRCNN did outperform existing methods, however its two stage anchor box based architecture was slow. To this end, 3DSSD was introduced as an efficient one stage model eliminating the computationally expensive feature propagation modules.

The architecture of 3DSSD was relatively simple compared to previous methods, as illustrated in Figure 2.4. At each stage, it used various sampling algorithms to select a number of a seed points from the input point cloud as input for the set abstraction layer. This included $D-FPS$, which is simply vanilla Farthest point sampling, and $F-FPS$ or Feature-Farthest point sampling, which selected points based on the spatial distance and semantic feature distance as the criterion in FPS.

$$C(A, B) = \lambda L_d(A, B) + L_f(A, B)$$

where $L_d(A, B)$ and $L_f(A, B)$ represent L^2 XYZ distance and L^2 feature distance between two points and λ is the balance factor.

At the final stage, the resulting feature vectors are fed into two MLPs to predict the class and parameters of bounding boxes. The entire architecture is shown in Figure 2.4

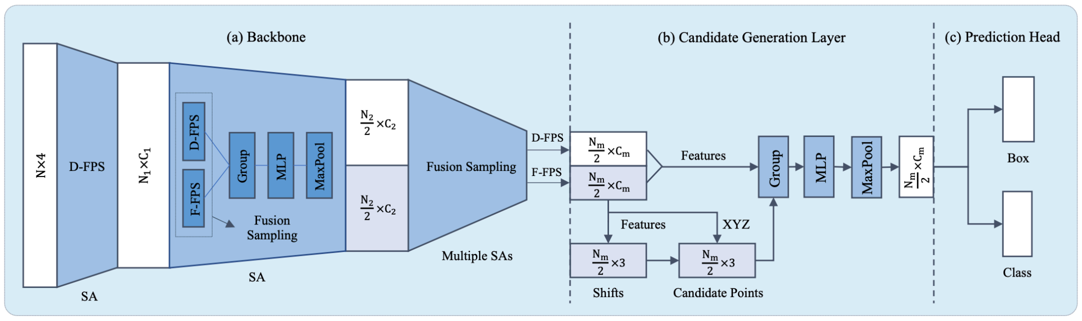


Figure 2.4: 3DSSD's single stage detector architecture. Image taken from [10].

3DSSD outperformed previous single stage methods, and have similar performance to the top performing two stage detector. This is while being 2.6 times faster than PointRCNN, and 2.1 times faster than STD, the best performing two stage detector at the time.

2.1.5 IA-SSD

IA-SSD was introduced by Zhang et al. in the 2022 paper “Not All Points Are Equal: Learning Highly Efficient Point-Based Detectors for 3D LiDAR Point Clouds.”

IA-SSD’s main improvement upon 3DSSD’s single stage architecture by introducing a new centroid aware sampling strategy during the set abstraction layers.

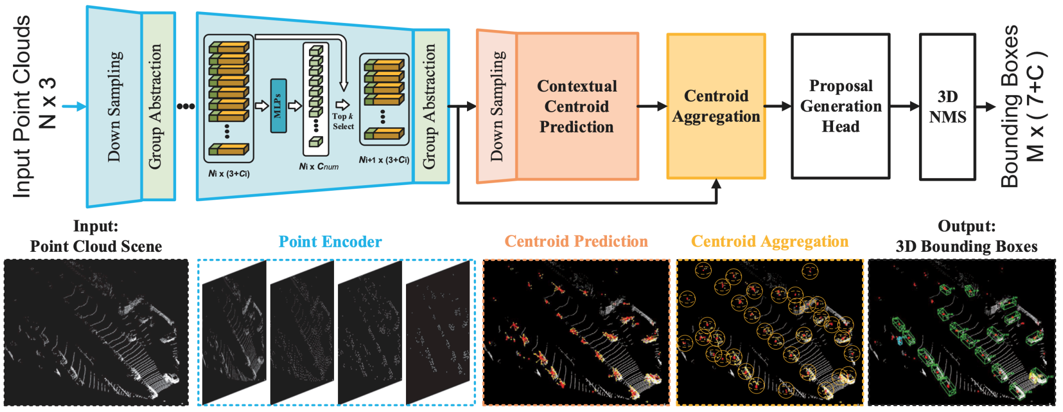


Figure 2.5: IASSD’s single stage detector architecture. Image taken from [3].

This is accomplished by utilizing a soft point mask during training

$$Mask_i = \sqrt{\frac{\min(f^*, b^*)}{\max(f^*, b^*)} \times \frac{\min(l^*, r^*)}{\max(l^*, r^*)} \times \frac{\min(u^*, d^*)}{\max(u^*, d^*)}} \tag{2.1}$$

where $f^*, b^*, l^*, r^*, u^*, d^*$ represent the distance of a point to the 6 surfaces (front, back, left, right, up and down) of the bounding box, respectively. This mask is used to assign higher weight towards points inside a bounding box via a weighted cross entropy loss to train a MLP to output confidence values. These confidence values outputted by the MLP are then used during inference to keep the top k points, therefore performing a downsampling operation with preference towards points closer to object centroids.

With this centroid-aware downsampling strategy, IASSD removes the use of $F - FPS$ of 3DSSD, which were still computationally intensive. IASSD has maintained competitive performance with both two stage and single stage methods, and most notably being much more efficient than all previous methods. For example, it is 5.9 times faster than PointRCNN.

2.2 Point Based Transformer Methods

The following section provides an overview of several related works that explore transformer based architectures for point cloud based detection.

2.2.1 Point Cloud Transformer

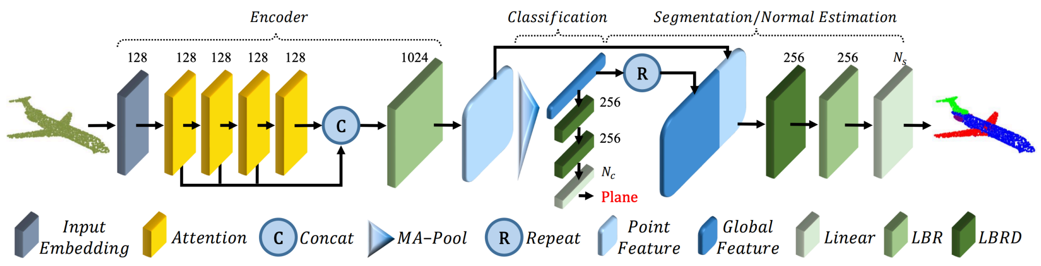


Figure 2.6: PCT architecture. Image taken from [11].

The Point Cloud Transformer (PCT) is a novel framework designed specifically for point cloud learning. It successfully adapts the transformer architecture from natural language processing to handle unordered point sets effectively and efficiently. The architecture of PCT is illustrated in Figure 2.6 [11]. The key contributions of PCT can be summarized as follows:

Coordinate-based Input Embedding: PCT introduces a coordinate-based input embedding module that incorporates raw positional encoding and input embedding. Since each point in a point cloud has unique coordinates representing its spatial position, this module generates distinguishable features that allow the model to process unordered point sets.

Optimized Offset-Attention Module: PCT improves upon the original self-attention mechanism with an optimized offset-attention module. This approach replaces the attention feature with the offset between the input of the self-attention module and the attention feature. The offset-attention module offers two advantages: it uses relative coordinates, which are more robust due to varying absolute coordinates caused by rigid transformations, and it is akin to a Laplace process, as the point cloud is treated as a graph with a float adjacency matrix as the attention map.

Neighbor Embedding Module: To enhance the point embedding process, PCT incorporates a neighbor embedding strategy. The attention mechanism is effective in capturing global features but may overlook local geometric information. The neighbor embedding module addresses this issue by considering attention between local groups of points containing semantic information, rather than individual points.

PCT demonstrated state-of-the-art performance on shape classification, part segmentation, semantic segmentation, and normal estimation tasks. However, it was not used for object detection.

2.2.2 3DETR

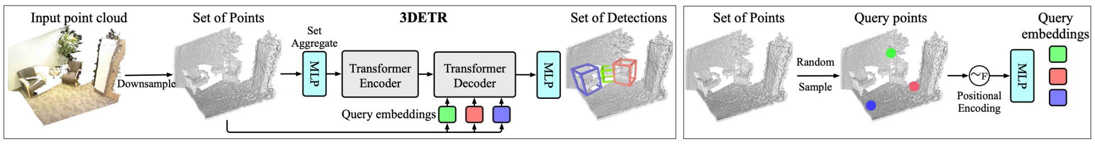


Figure 2.7: Left: 3DETR architecture, the Transformer encoder produces a set of per-point features using multiple layers of self-attention. The point features and a set of 'query' embeddings are input to the Transformer decoder that produces a set of boxes. The predicted boxes are matched to the ground truth and optimized with a set loss.

3DETR [2] is an end-to-end Transformer-based model for 3D object detection in point clouds, offering a competitive and conceptually simple alternative to specialized architectures that rely on hand-tuned hyperparameters and 3D-specific operators. By employing standard Transformers with non-parametric queries and Fourier positional embeddings, 3DETR is able to process unordered 3D point cloud data effectively, casting 3D object detection as a set-to-set problem. The model's architecture is illustrated in Figure 2.7.

The 3DETR model follows a general encoder-decoder structure similar to both its 2D counterpart DETR [32] and the vanilla transformer architecture [29]. It replaces the PointNet++ set abstraction layer with a standard Transformer applied directly on point clouds, and adapts the parallel decoding strategy from DETR with Transformer layers to suit 3D detection. Key modifications include non-parametric query embeddings and Fourier positional embeddings.

Experiments were only conducted on standard indoor 3D detection datasets ScanNetV2[33] and SUN RGB-D [34], and efforts to reproduce 3DETR on outdoor datasets like KITTI [35] have met little success. In a recent paper by He et al.[36], they claim that "3DETR present a promising solution by computing self-attention on a reduced set of seed points, this solution is only applicable to indoor scenes, where the point clouds are relatively dense and concentrated."

2.2.3 Pointformer

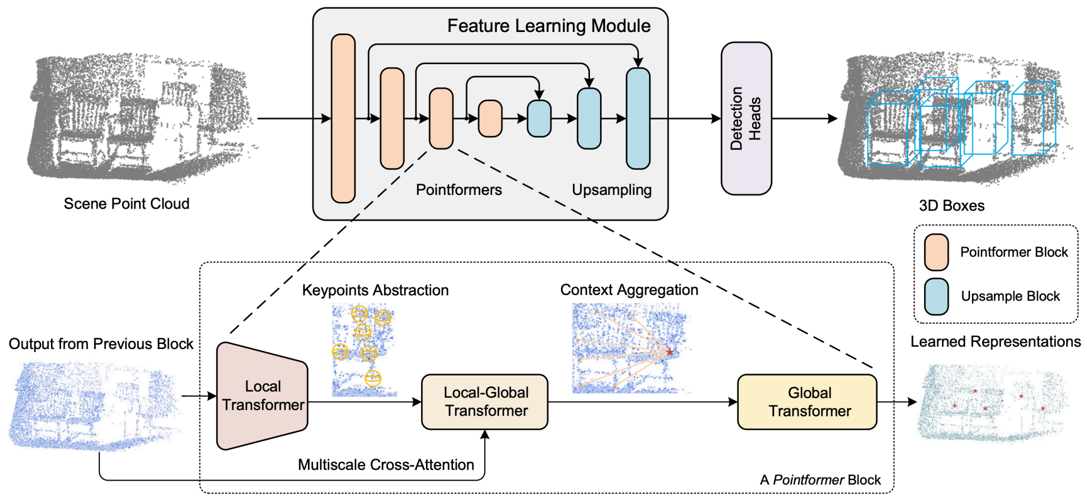


Figure 2.8: Pointformer architecture. Image taken from [12].

Pointformer [12] is another transformer-based backbone, unlike the previous two methods however, pointformer was tested on the outdoor self driving dataset KITTI [35]. Pointformer utilizes various modified transformer blocks they denote by pointformer blocks, as follows:

Local Transformer (LT) Module: The LT module models interactions among points within a local region, learning context-dependent region features at an object level. This enables the Pointformer to capture local dependencies and improve feature learning for scenes with multiple cluttered objects.

Local-Global Transformer (LGT): The LGT integrates local features with global features from higher resolutions, allowing the Pointformer to capture both local and global dependencies effectively.

Global Transformer (GT) Module: The GT module learns context-aware representations at the scene level, further enhancing the Pointformer's ability to process 3D point clouds effectively.

Chapter 3

Methodology

In the follow chapter, we will introduce the dataset and framework used to develop and test our models, the metrics used to evaluation detection models and the several attention free and attention based methods we introduce. Finally we also outline the training details of the methods.

3.1 Dataset

3.1.1 Popular datasets

One of the most popular datasets used for self-driving research is the KITTI Vision Benchmark Suite, which was developed by the Karlsruhe Institute of Technology and the Toyota Technological Institute at Chicago. The KITTI dataset includes a range of data types, including high-resolution color images, grayscale stereo images, 3D point clouds captured by a Velodyne LiDAR sensor, and GPS/IMU data. The dataset contains 7481 training images and 7518 test images, with a total of 80,256 labeled objects in the training set and 80,408 labeled objects in the test set. The labeled objects include cars, pedestrians, and cyclists, and are divided into eight classes: car, van, truck, pedestrian, person sitting, cyclist, tram, and miscellaneous. The KITTI dataset has been widely used to train and test object detection, tracking, and segmentation algorithms for autonomous driving [35].

Another notable dataset is the nuScenes dataset [4], which was developed by autonomous vehicle technology company nuTonomy (now part of Aptiv). The nuScenes dataset includes 1000 scenes captured in Boston and Singapore, with each scene consisting of 20 seconds of sensor data captured by a range of sensors, including six high-resolution cameras, one LiDAR sensor, and five radar sensors. The dataset contains 1.4 million images, 400,000 LiDAR scans, and 1.1 million 3D annotated bounding boxes across 23 object classes. In addition, the nuScenes dataset includes information about the weather, lighting conditions, and traffic flow for each scene. The nuScenes dataset has been widely used for a variety of tasks, including object detection, tracking, and behavior prediction.

Another notable dataset is the Waymo Open Dataset [37], which was developed by Alphabet subsidiary Waymo. The dataset was captured with the following sensors by Waymo's autonomous vehicles, including five high-resolution cameras and 5 LiDAR sensors. The dataset contains over 12.6 million 3D annotated bounding boxes with tracking ID across 28 different classes. The Waymo Open Dataset has been used to train and test a range of object detection and tracking algorithms, as well as for other tasks such as sensor fusion and semantic segmentation.

3.1.2 Selected dataset

Although all the previous datasets have been popular for training state of the art architectures, they cannot be used in this case. 4D radar only became accessible in the past two years, and therefore none of these datasets have 4D radar data. Recently, the View Of Delft (VoD) Dataset [1] was released. The VoD dataset is

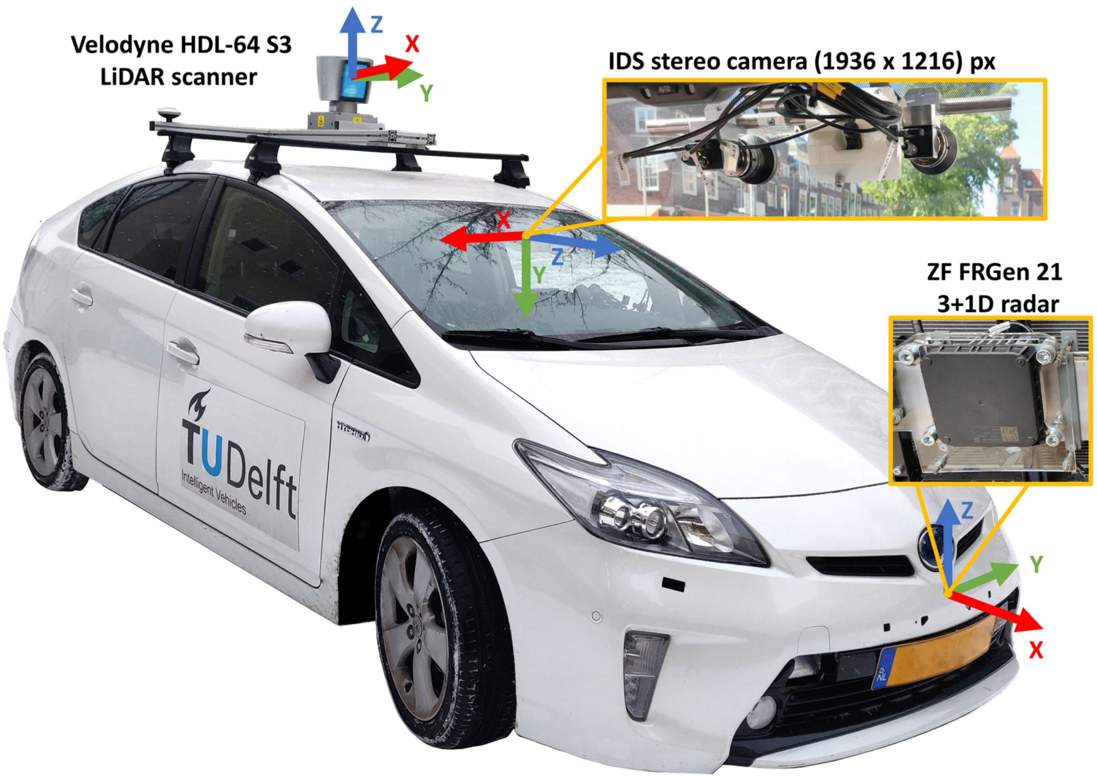


Figure 3.1: Vehicle and sensors used for the View of Delft dataset

the first publicly available dataset with 4D radar data. The size of the dataset is comparable to KITTI, with 8600 frames and 13 annotated classes collected in the city streets of the Dutch city Delft. While there exists two other datasets with 4D radar data (TJ4DRadSet[38], KAIST-Radar[39]), they are not currently publicly accessible yet and therefore not usable for now. As a result, all of the investigation in this thesis will be conducted using the VoD dataset.

3.2 Framework used

The foundation of our experiments is the OpenPCDet framework [40], an open-source project developed by the OpenMMLab team, which provides a comprehensive framework for 3D object detection from point clouds.

OpenPCDet is a versatile and flexible library designed to facilitate the implementation, training, and evaluation of 3D object detection models that utilize point cloud data. Point clouds are sets of points in a 3D coordinate system, usually obtained from devices such as LiDAR sensors or stereo cameras. OpenPCDet provides a modular and extensible platform that integrates various state-of-the-art 3D object detection methods, making it an ideal choice for our experiments. Furthermore, the authors of the baseline model IASSD also used OpenPCDet, therefore developing upon IASSD in OpenPCDet is a natural choice.

The library supports many prominent 3D object detection algorithms, including PointPillars [41], SECOND , and Part-A^2 Net, among others. Furthermore, it provides extensive pre-processing, data augmentation, and evaluation tools that enable seamless experimentation and customization. The library is implemented in Python and built on top of the PyTorch deep learning framework, allowing for efficient GPU acceleration during model training and inference.

3.2.1 Framework structure

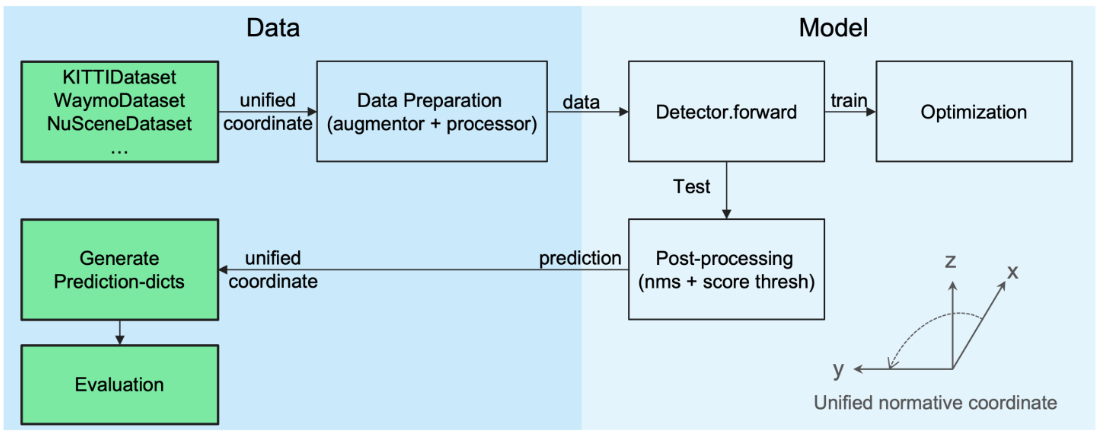


Figure 3.2: Design pattern of OpenPCDet

Figure 3.2 illustrates the design pattern of OpenPCDet. There currently exists many popular dataset formats, but the three most common follow the format of KITTI, Waymo, and NuScenes. The VoD dataset follows the format of KITTI, where the data for the lidar sensor is stored as follows:

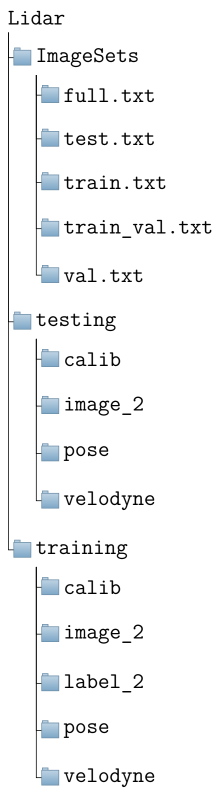


Figure 3.3: Directory tree of the View of Delft Dataset

The point clouds themselves are stored as binary files under the `velodyne` folder, images under the `image_2` folder, calibration files containing the data to calculate the transformation matrices from one reference frame to another (e.g. LiDAR to camera, radar to LiDAR), and the ground truth bounding boxes are in the `label_2` folder in a text file where each line represents an object,

OpenPCDet then converts the data into a unified coordinate system, ready for data preparation. The framework has many built-in data processing and augmentation functions, a few of them are listed below.

Values	Name	Description
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare'
1	truncated	Not used, only there to be compatible with KITTI format.
1	occluded	Integer (0,1,2) indicating occlusion state: 0 = fully visible, 1 = partly occluded 2 = largely occluded.
1	alpha	Observation angle of object, ranging $[-\pi..\pi]$
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x, y, z in camera coordinates (in meters)
1	yaw	Rotation around -Z axis of the LiDAR sensor $[-\pi..\pi]$
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

Table 3.1: KITTI format object label

Data processing

- Voxelization: Transforming raw point cloud data into a structured 3D grid representation by dividing the point cloud into voxels, which can help improve the efficiency of the detection process.
- Masking points: Removing points outside a range
- Shuffling points: Shuffling the order of points
- Sampling points: Sampling n points from each input.

Data Augmentation:

- Point cloud rotation: Rotating the point cloud around a specific axis to generate new orientations of the data and improve model robustness.

- Point cloud scaling: Scaling the point cloud data by a random factor to simulate different object sizes and improve model generalization.
- Point cloud flipping: Flipping the point cloud along one or multiple axes to generate new views of the data.
- Random sampling: Randomly sampling points from the point cloud to generate a fixed-size input for the model, which can help control the computational complexity.
- Ground Truth Sampling: Placing additional ground truth boxes and the containing points for each object class.

Such augmentation methods are frequently used during training of object detection models like [10][9][3].

3.3 Evaluation Metrics

3.3.1 Intersection over Union (IoU)

In the realm of 3D object detection, evaluating the performance and accuracy of a model is crucial for understanding its effectiveness in real-world scenarios. This section provides an overview of commonly used evaluation metrics for 3D object detection models, including 3D Intersection over Union (IoU), and Bird's Eye View (BEV) IoU.

3D Intersection over Union (IoU) is a widely-used evaluation metric for 3D object detection. It quantifies the overlap between the predicted and ground-truth 3D bounding boxes, providing an intuitive measure of the model's accuracy. The 3D IoU is defined as the ratio of the volume of the intersection between the two bounding boxes to the volume of their union. The 3D IoU can be expressed as:

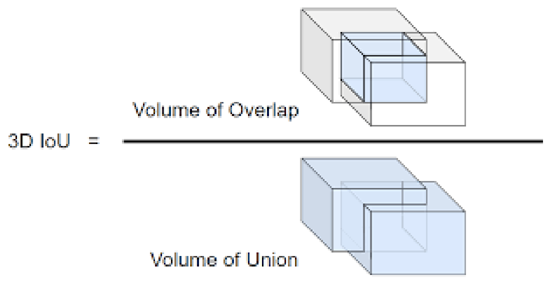


Figure 3.4: Mathematical expression of 3D IoU. Image taken from [13]

A higher IoU value indicates a better alignment between the predicted and ground-truth bounding boxes, with 1 being a perfect match and 0 indicating no overlap. The 3D IoU metric is particularly useful for measuring the localization accuracy of a 3D object detection model, as it considers the spatial dimensions and orientation of the bounding boxes.

The Bird's Eye View (BEV) IoU is another evaluation metric that assesses the performance of 3D object detection models. It focuses on the 2D projection of the 3D bounding boxes onto the ground plane, providing a top-down perspective. This metric measures the overlap between the predicted and ground-truth 2D bounding boxes in the horizontal plane, disregarding the height information.

To calculate the BEV IoU, the following formula is used:

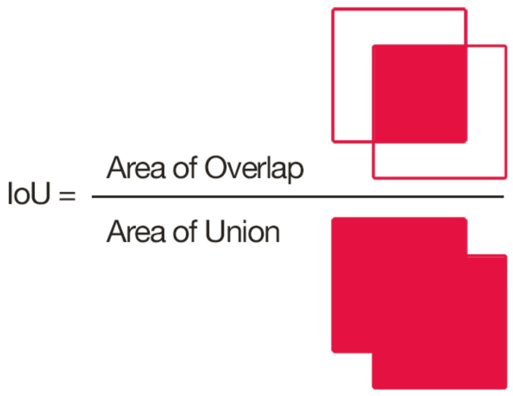


Figure 3.5: Mathematical expression of BEV IoU. Image taken from [14]

Similar to the 3D IoU, a higher BEV IoU value signifies a better alignment between the predicted and ground-truth bounding boxes, with 1 indicating a perfect match and 0 representing no overlap. BEV IoU is particularly relevant in scenarios where the primary focus is on the horizontal localization accuracy,

3.3.2 Average Precision

We can now understand the context of using average precision (AP) as an evaluation metric for object detection models. To calculate AP, we utilize precision and recall

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

where true positives (TP) are the number of instances correctly predicted as positive by the model, false positives (FP) are the number of instances incorrectly predicted as positive by the model, and false negatives (FN) are the number of instances incorrectly predicted as negative by the model. In the case for 3D object detection, TP are predictions that have a 3D IoU with the ground truth of greater than some threshold.

Suppose our model outputted n predictions, the predictions are first sorted by their confidence value, then Precision_i and Recall_i are calculated for $i = 1, \ldots, n$, where Precision_i and Recall_i are the metrics calculated with the top i confident predictions. The precision values can then be plotted against the recall values in what is commonly called the precision-recall (PR) curve. An example of a PR curve is illustrated in Figure 3.6.

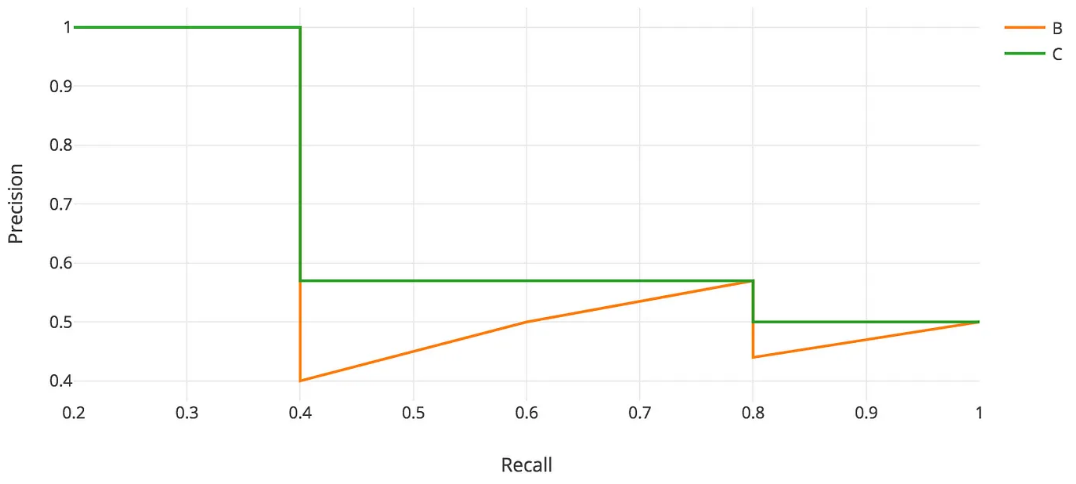


Figure 3.6: Orange curve: raw precision-recall points . Green curve: Interpolated precision recall points. Image taken from [15]

The raw PR curve is then smoothed out by replacing all precision values p at a recall value r with the maximum precision for all recall values greater than r. Then finally, the convention is to select 11 evenly spaced out points on this green curve and average the precision values to yield the AP.

For all experiments, we calculate the AP for the classes car, pedestrian, and cyclist with an IoU threshold of 0.5,0.25,0.25 respectively, unless stated otherwise, in accordance to the authors of the VoD dataset [1]. Finally, mean average precision (mAP) is a metric used to demonstrate a model's overall performance across all classes. mAP is calculated by taking the average of the AP of each class.

$$mAP = \frac{1}{n_{\text{classes}}} \sum_{i}^{n_{\text{classes}}} AP_i$$

3.4 Baseline Models

3.4.1 Transformer baseline

First to address research question 1, we would like to test whether it is possible to employ an encoder-decoder network similar to the vanilla transformer model. Since 3DETR most resembles this, we first determine whether 3DETR can be adapted for this purpose.

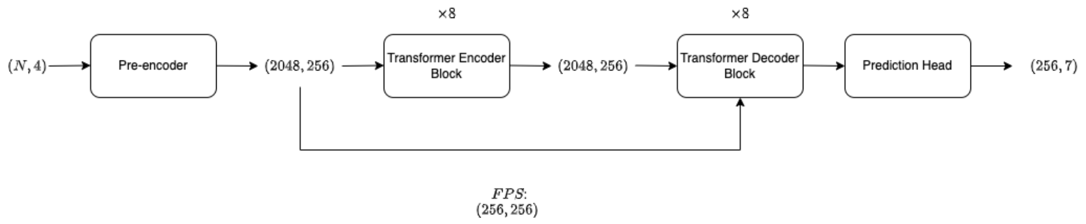


Figure 3.7: 3DETR

Figure 3.7 illustrates the 3DETR architecture in more detail. The input point cloud is processed by the pre-encoder which downsamples the point cloud to 2048 points through a shared MLP. Next, the points are passed through eight normal transformer encoder blocks. 256 seed points are then sampled from the 2048 points via FPS and used as the input to the decoder block. The seed points and the encoder output are then used as input for the decoder block, similarly repeated eight times before regressing into bounding boxes.

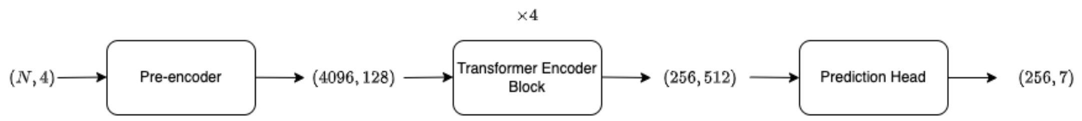


Figure 3.8: Encoder only 3DETR

Furthermore, we also test an encoder only model by removing the decoder blocks from the 3DETR module and reducing the number of encoder blocks.

After each encoder block, we downsample the points and pass the features through an MLP to increase the feature dimension. The number of points and feature dimension is chosen to be $(4096, 128) \rightarrow (1024, 128) \rightarrow (512, 128) \rightarrow (256, 512)$ to closely resemble the number of points in each set abstraction layer in IASSD as illustrated in Figure 3.9.

3.4.2 IASSD baseline

In addition to adapting 3DETR to our dataset, we will also test several methods by modifying an existing state of the art architecture. The model chosen is IA-SSD, which is currently the most efficient single stage detector with competitive performances to two stage detectors. Figures 3.9 and 3.10 illustrate the architecture of IASSD trained on LiDAR and radar inputs denoted as IASSD-L and IASSD-R respectively. The main difference is the number of points sampled at each set abstraction layer.

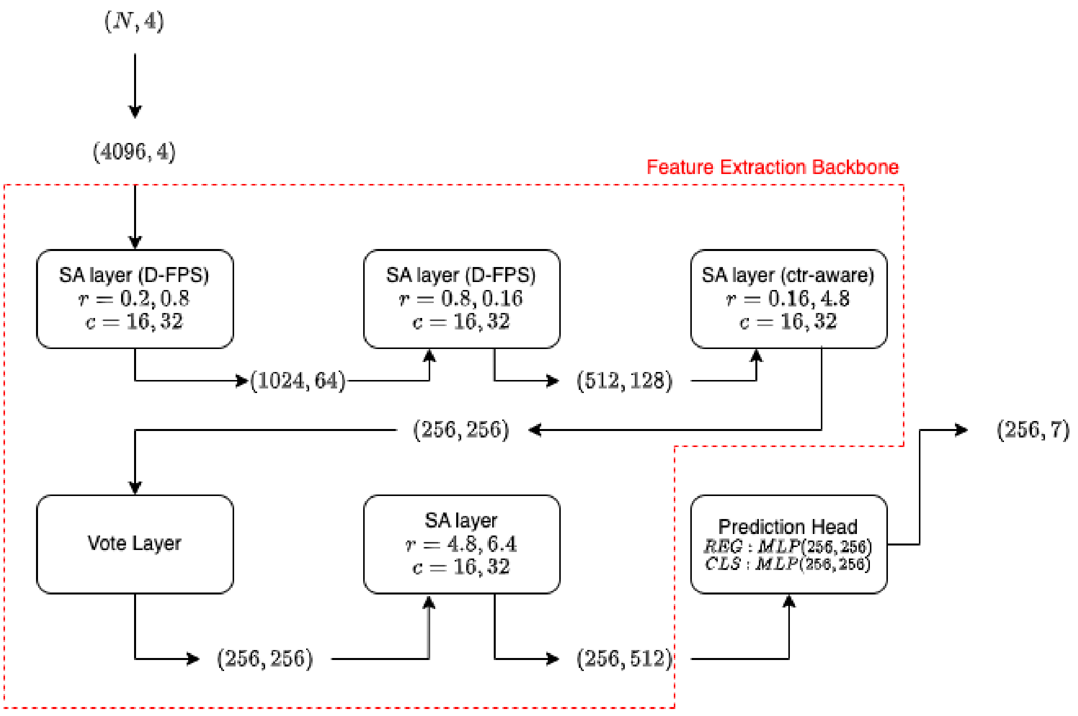

$(N, 4)$
$(4096, 4)$
Feature Extraction Backbone
SA layer (D-FPS) $r = 0.2, 0.8$ $c = 16, 32$
SA layer (D-FPS) $r = 0.8, 0.16$ $c = 16, 32$
SA layer (ctr-aware) $r = 0.16, 4.8$ $c = 16, 32$
$(1024, 64)$
$(512, 128)$
$(256, 256)$
$(256, 7)$
Vote Layer
SA layer $r = 4.8, 6.4$ $c = 16, 32$
Prediction Head $REG : MLP(256, 256)$ $CLS : MLP(256, 256)$
$(256, 256)$
$(256, 512)$

Figure 3.9: Outline of the IASSD-L architecture

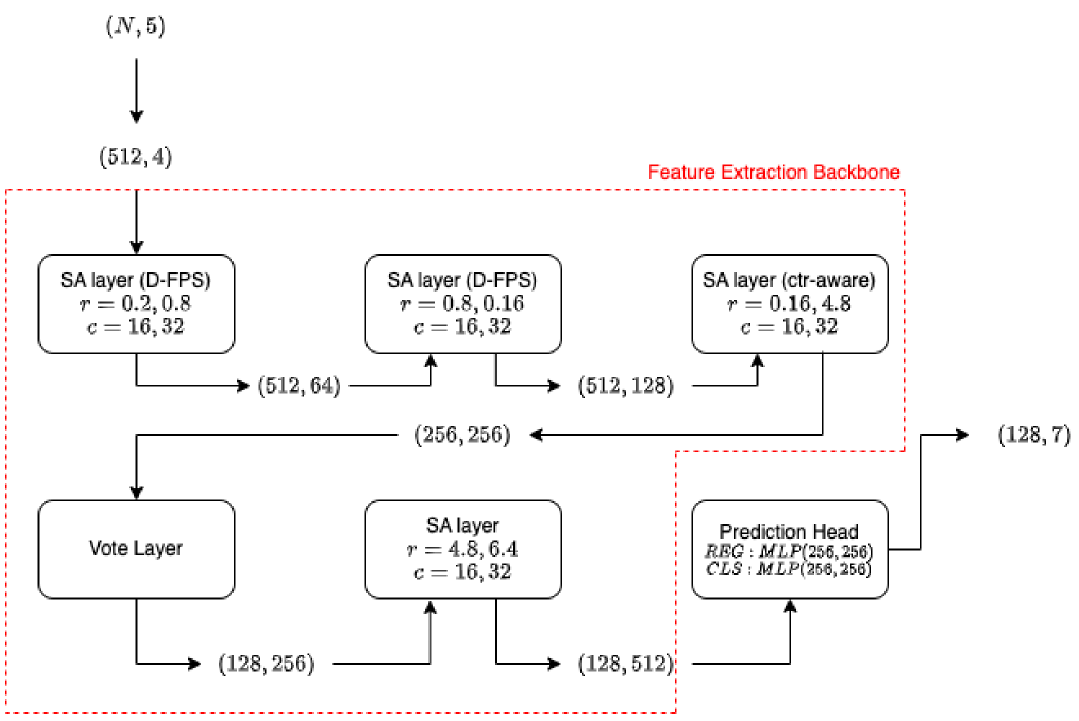

(N, 5)
(512, 4)
Feature Extraction Backbone
SA layer (D-FPS) r = 0.2, 0.8 c = 16, 32
SA layer (D-FPS) r = 0.8, 0.16 c = 16, 32
SA layer (ctr-aware) r = 0.16, 4.8 c = 16, 32
(512, 64)
(512, 128)
(256, 256)
(128, 7)
Vote Layer
SA layer r = 4.8, 6.4 c = 16, 32
Prediction Head REG : MLP(256, 256) CLS : MLP(256, 256)
(128, 256)
(128, 512)

Figure 3.10: Outline of the IASSD-R architecture

IASSD-L during training, each LiDAR point cloud is first downsampled to 4096 points via Furthest Point Sampling. The point cloud is then fed into the first set abstraction layer, resulting in 1024 points each with feature dimension of 64. This is then repeated twice more to 256 points with feature dimension of 256 before passing through a vote layer to shift points towards potential centroids, before one more set abstraction layer to yield 256 points with feature dimension of 512. These 256 points can be interpreted as 256 proposals for potential objects. They are then fed into the prediction head which consists of two MLP branches. The CLS branch predicts the class of each proposal, and the REG branch predicts the parameters of the bounding box, such as x, y, z, l, w, h, θ. These two branches combined output 256 bounding boxes, which during training is fed into various loss functions. During inference, post processing is then applied, like non-maximal suppression, to yield the final predicted bounding boxes. Training process is identical for IASSD-R except for number of downsampled points.

More generally, we can describe the consecutive set abstraction layers as the feature extraction backbone as outlined in red in Figures 3.9 and 3.10. As the layers in aggregate take in a set of point clouds, and output a set of features for the prediction head.

3.5 Attention Free Methods

Compared to other multimodal fusion problems, such as text with image, audio with video, where the representation of the modalities the problem of fusing LiDAR and radar sensor information may seem ostensibly more straightforward given that both LiDAR and 4D radar sensor information is represented as point clouds. Contrasting to other common multimodal problems with text being a sequence of words or characters, images are 2D grids of pixels with color channels (e.g., RGB), and audio is a time series of continuous or discrete samples. This heterogeneity in data representation makes it challenging to develop a unified framework for processing and combining the data.

Although both data from both sensors are captured in the form of a point cloud, fusing information is still no easy task given the characteristics of point cloud processing. Thus, the following methods we introduced will be explored.

3.5.1 Point Cloud Concatenation

A canonical approach that one may hope would work is to preprocess both cloud points and combine them into one. Let $\mathcal{R} \in \mathbb{R}^{3+2}$ be the radar point cloud and $\mathcal{L} \in \mathbb{R}^{3+1}$ be the LiDAR point cloud, then we apply the map points from both modalities to a common feature dimension using $f_r : \mathcal{R} \to \mathbb{R}^6$ and $f_l : \mathcal{L} \to \mathbb{R}^6$, where the mapping functions are defined as.

$$f_l\left(\begin{bmatrix} x & y & z & I \end{bmatrix}^T\right) = \begin{bmatrix} x & y & z & I & 0 & 0 \end{bmatrix}^T$$
$$f_r\left(\begin{bmatrix} x & y & z & RCS & v_{doppler} \end{bmatrix}^T\right) = \begin{bmatrix} x & y & z & 0 & RCS & v_{doppler} \end{bmatrix}^T$$

We can then concatenate the sets of points, thus our input point cloud $\mathcal{P}$ becomes

$$\mathcal{P} \in \mathbb{R}^6 := \{f_l(p) \mid \forall\, p \in \mathcal{L}\} \cup \{f_r(p) \mid \forall\, p \in \mathcal{R}\}$$

The motivation behind this approach is not dissimilar to audio-image tasks where the audio data is first transformed into a spectrogram, and concatenated along the channel axis of the image. Another view of this approach is simply that we are padding to match the feature dimensions and concatenating the set of points.

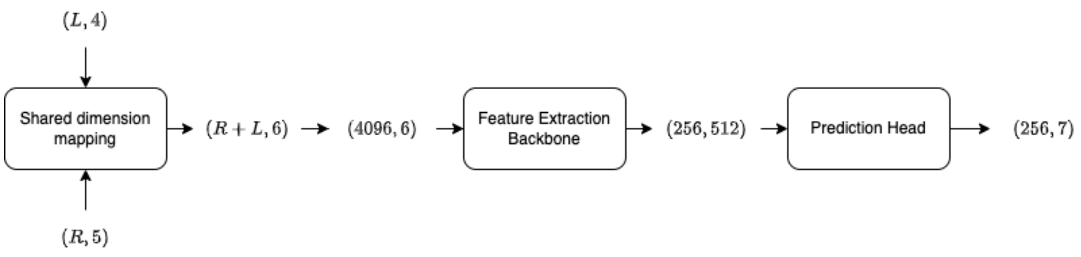


Figure 3.11: Point cloud concatenation architecture, L denotes the number of LiDAR points and R denotes the number of radar points for a training example

By transforming both LiDAR and radar data into a common feature space, we can more effectively combine the two modalities into a single input. This approach may help in leveraging the strengths of both LiDAR and radar data, providing a more comprehensive representation of the environment. As a result, the performance of 3D object detection models could be improved, as the models would have access to richer feature information from both LiDAR and radar sources while maintaining similar computational costs as only one backbone is needed. The modifications to IASSD to test this method are illustrated in Figure 3.11, where we only need one additional lightweight layer before the backbone.

3.5.2 Feature interpolation

Another reasonable approach can be derived from examining the drawbacks of both modalities. LiDAR provides robust and detailed geometric information, but is unable to provide any additional information about the collected data points. I.e. the point cloud of a person moving at 50mph would appear the same as someone not moving at all. Which the 4D radar sensor is able to provide, as it contains information about the material (RCS) and the velocity (Doppler velocity) of collected points. Thus the question arises: would the hypothetical optimal sensor be one that is able to capture a dense and feature rich point cloud? I.e. would the ideal sensor be one that captures points with features $[x, y, z, Intensity, RCS, v_{doppler}]$?

To test this hypothesis, we apply Algorithm 1 as outlined below to approximate this hypothetical ideal point cloud by interpolating radar features to neighboring LiDAR points.

We first define the following helper functions. `knn(K,l,R)` is the standard knn neighbours, that samples K nearest neighbours of point l within a radius R in the radar point cloud $\mathcal{R}$. It returns a subset of $\mathcal{R}$ with size K.

$$\texttt{knn} : \mathcal{R} \to \mathcal{R}' \text{ where } \mathcal{R}' \subseteq \mathcal{R} \text{ and } |\mathcal{R}'| = K$$

`average_features(k_radar_points)` takes in a set of radar points, and av-

Algorithm 1 Feature interpolation

```
Require: $\mathcal{L} \in \mathbb{R}^{3+1}$ set of LiDAR points
Require: $\mathcal{R} \in \mathbb{R}^{3+2}$ set of radar points
  $R \leftarrow r$                                   ▷ $r$ is sampling radius
  $K \leftarrow k$                                   ▷ $k$ is number of neighbours
  $\mathcal{L}_{new} \leftarrow []$
  for $l$ in $\mathcal{L}$ do
      k_radar_points ← knn($K, R, l, \mathcal{R}$)
      k_features← average_features(k_radar_points[3 :])
      $l$ ←concat($l$,k_features)
      $\mathcal{L}_{new}$.append(l)
  end for
  return $\mathcal{L}_{new}$
```

erages the features of the points excluding x, y, z, its coordinates. I.e. the function averages the RCS and $v_{doppler}$ values of the set of points.

$$\texttt{average\_features} : \mathcal{R}' \rightarrow \mathbb{R}^2$$

Finally, `concat(l,k_features)` simply concatenates the averaged radar features to the LiDAR point l. I.e. it maps the LiDAR point from $(x, y, z, Intensity)$ to $(x, y, z, Intensity, \overline{RCS}, \overline{v_{doppler}})$.

$$\texttt{concat} : \mathcal{L} \rightarrow \mathbb{R}^6$$

The algorithm begins by initializing the input LiDAR points, $\mathcal{L}$, and radar points, $\mathcal{R}$. It then proceeds to iterate through each LiDAR point, finding the K nearest radar points to each LiDAR point using the k-nearest neighbors (knn) method. The average of the radar point features is then computed, and the LiDAR point is augmented with these interpolated features. This updated LiDAR point is then appended to the new LiDAR point set, $\mathcal{L}new$. Once all LiDAR points have been processed, the updated point set, $\mathcal{L}new$, is returned.

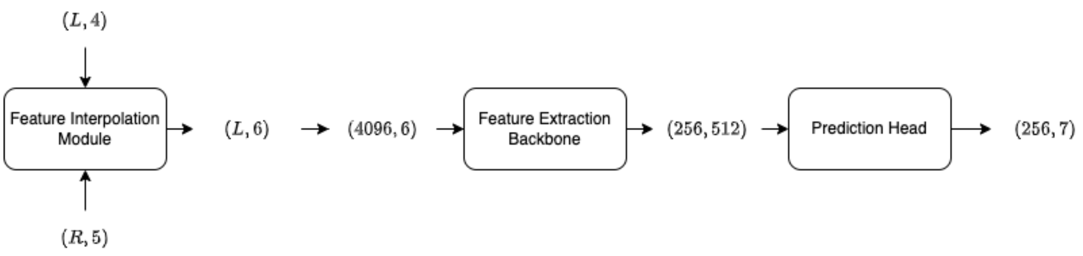


Figure 3.12: Feature interpolation architecture, L denotes the number of LiDAR points and R denotes the number of radar points for a training example

The architecture used to test this method is illustrated in Figure 3.12, like directly concatenating the point clouds, this method is lightweight and only requires one additional module prior to the feature extraction backbone. This method allows us to examine the potential benefits of combining the strengths of both LiDAR and radar data in a single, dense point cloud. By incorporating the interpolated radar features into the LiDAR data, the resulting point cloud may provide a more comprehensive representation of the environment, potentially improving the performance of 3D object detection models.

3.5.3 Feature fusion with MLP

Given how sensor fusion is typically categorized into early and late stage [42], we will now introduce a late stage method to test late stage fusion.

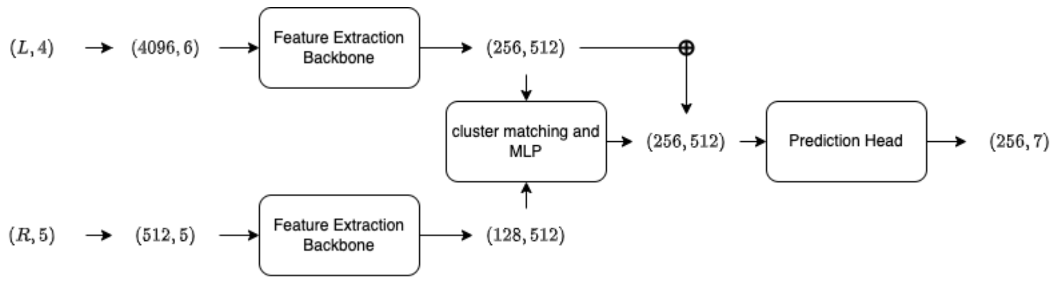


Figure 3.13: Late stage feature fusion, L denotes the number of LiDAR points and R denotes the number of radar points for a training example. $\oplus$ represents a skip connection

As illustrated in Figure 3.13, we first employ two separate feature extraction backbones, one for LiDAR and one for radar respectively to extract features for each modality. Next, we match each of the LiDAR proposals to its nearest radar proposal and concatenate the features before passing them through a multilayer layer perceptron with a skip connection. The resulting features have locally relevant features from the opposing modality, and hence contains more information compared to the baseline single modality method.

While late stage feature level fusion may have its advantages, one clear downside is the additional computational cost required to train the additional feature extraction backbone for the second modality.

3.6 Attentional Methods

Next, we dicuss the methods used to test whether introducing the attention mechanism can benefit multimodal fusion. All such methods will be tested on the final set abstraction layer of the feature extraction backbone. This is due to the fact that literature shows evidence that the attention mechanism benefits when the input feature sequence has higher dimension [29].

The three attention based methods will be tested using the following layer that we will now introduce, the **m**ulti**m**odal **s**et **a**bstraction (MMSA) layer. All point based object detection models that utilize the pointnet++ set abstract layer benefit from the fact that throughout each layer, we aggregate local point features through local sampling and grouping. Therefore with an additional modality, we must take into consideration the importance of locality. Thus, the key idea behind the MMSA layer is first matching each LiDAR cluster to a radar cluster, as illustrated in Figure 3.14. If both of the LiDAR and radar clusters are close together in the scene, one can expect that the points in both modalities to belong to the same foreground or background object.

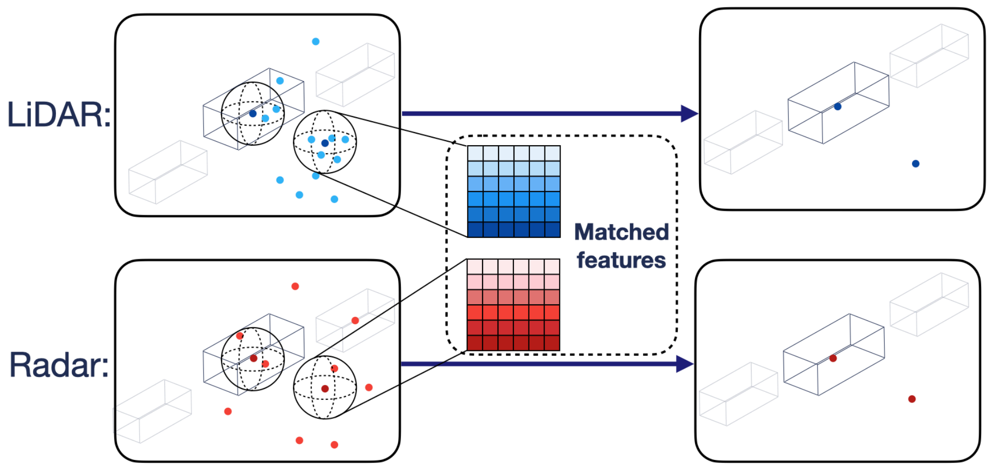


Figure 3.14: cross modality cluster matching, the darker colored points are the points selected via the sampling procedure. For each point, neighbouring points within some radius are sampled.

Since we will be replacing the final set abstract layer with the proposed MMSA layer, the features at this stage are physically close to the final prediction proposals. Thus we should expect an effective mechanism to be able to combine the LiDAR and radar features.

3.6.1 Single Cross Attention Layer

The most direct method to incorporate attention is to apply cross attention on the matched features. Cross attention as originally proposed by Vaswani et al. [29] assigned one set of feature vectors as the query, and the other set of features to be the key and value. Again, the motivation behind this approach is to allow the query and key vectors to create a mask that reweights each key vector as a linear combination of all key vectors.

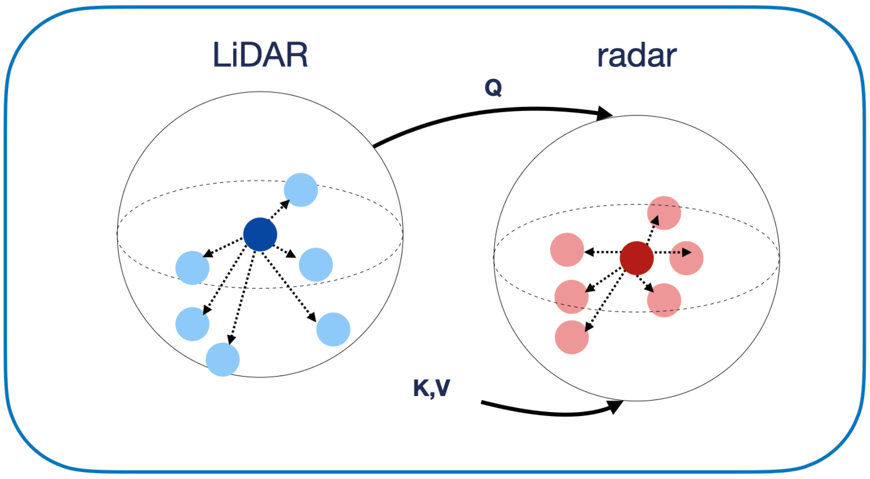


Figure 3.15: Query, key, value selection for cross attention applied to a single matched cluster

This is illustrated in Figure 3.15. Therefore after applying the cross attention layer, we will have a set of radar features reweighted by the LiDAR features. These features are then concatenated with the original LiDAR features before passing through a PointNet layer.

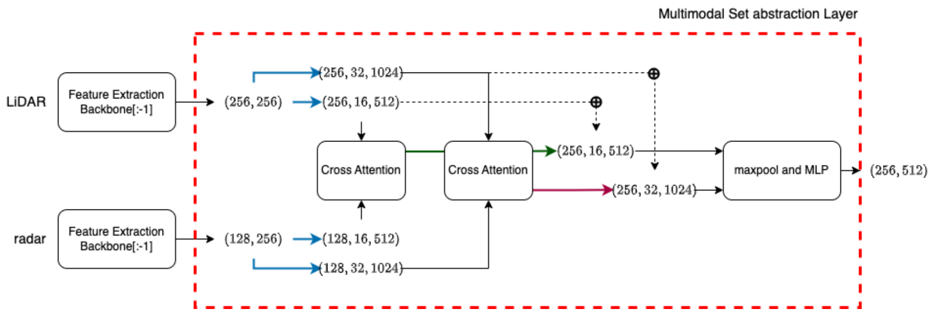


Figure 3.16: Operations in a MMSA layer. Blue arrows indicate multiscale grouping, dotted lines and ⊕ represent skip connection

Figure 3.16 illustrates how the MMSA layer behaves in more detail when replacing the final set abstraction layer. Points from both modalities are passed through their respective feature extraction backbones normally until the the final layer, which they are then inputted into the MMSA layer. As per pointnet++ convention, multiscale grouping is used to sampled points are different scales, yielding 16 and 32 clusters respectively. At each scale, cross attention is then applied with the LiDAR features being the query, and radar features being the key and value. Finally, the resulting attenuated features are then concatenated back to the original features before maxpooling and MLP to yield the final features. After understanding what the MMSA layer is doing, we can represent it like another module and therefore represent this proposed architecture as illustrated in Figure 3.17.

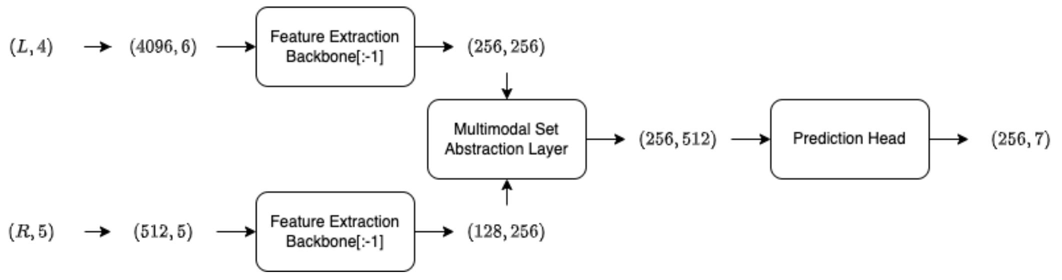


Figure 3.17: A higher level overview of the architecture utilizing a MMSA layer

3.6.2 Cross Attention Block

As discussed in the related works section, there has been some success in creating point based transformer models for object detection, but few specifically for outdoor scenarios like self driving. Pointformer was one architecture that was able to be applied to outdoor datasets like KITTI with competitive results. We therefore replace the single cross attention layer with a cross attention block, which is adapted from the Local-Global Transformer [12]. Where LiDAR and radar are the input instead of LiDAR points at different scales in the original Local-Global Transformer. The block is illustrated in Figure 3.18.

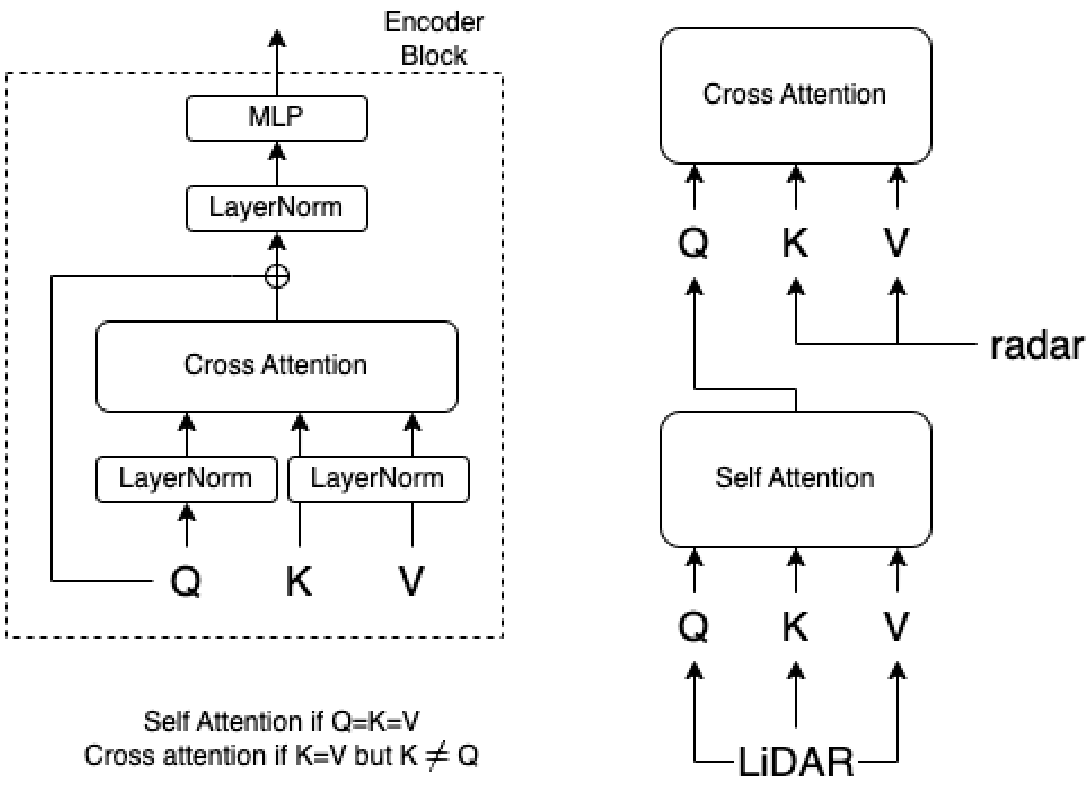


Figure 3.18: Left: Encoder block, which becomes self attention or cross attention, Right: Cross attention block which utilizes self attention and cross attention

3.6.3 Cross Attention Variations

Given the multimodal nature of our problem, it may be helpful to observe what approaches others are taking for different problems. For example, document and PDF analysis with deep learning is another challenging task due to the inherent nature of documents. Documents often consist of various elements, such as text, images, tables, and graphs, that are organized in diverse layouts and structures. Analyzing and understanding these elements require processing and extracting information from both the visual and textual modalities, making document analysis perhaps surprisingly a multimodal problem, despite ostensibly being a text based problem.

The multimodal aspect of document analysis necessitates the integration of computer vision and natural language processing techniques. Convolutional Neural Networks (CNNs) are often employed to extract visual features from document images, while NLP methods are utilized to process and extract textual information. This is similar to the multimodal problem of LiDAR and 4D radar, where complementary sensor modalities are combined to improve object detection and perception in autonomous systems.

One recent approach to tackling this problem was presented in the paper SelfDoc: Self-Supervised Document Representation Learning by [43], where optical character recognition was used to extract textual features and a document object detector using Faster R-CNN [44] to extract visual features. A transformer based method utilizing cross attention and self attention was then used to fuse the features together.

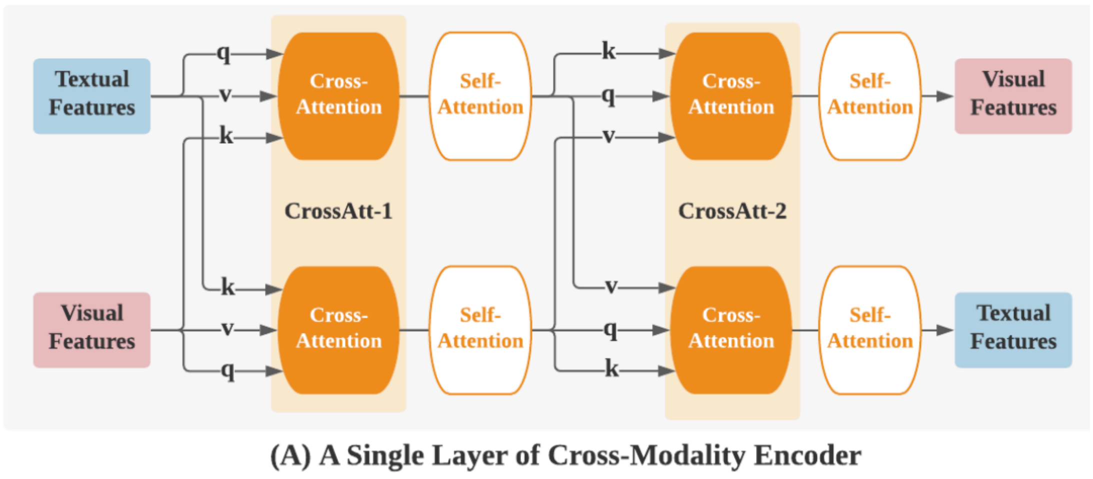


Figure 3.19: Cross-Modality encoder presented in the SelfDoc paper

However, the key difference between the cross attention used here compared to vanilla cross attention is the values of the query, key and values. Normally the key and value features come from one modality, while the query feature is from the other. In the cross-modality encoder, as illustrated in Figure 3.19, the query, key, and value features are selected differently. The different ways these values were selected are illustrated in Table 3.2, where A and B are sets of feature vectors from different modalities.

Scenario	Query (Q)	Key (K)	Value (V)
Normal Cross Attention	A	B	B
CrossAtt1	A	B	A
CrossAtt2	A	A	B

Table 3.2: Comparison of different choices of Q, K, and V for cross attention

In CrossAtt1, the attention scores between A and B is used to reweight A. In the context of LiDAR and radar features, where A represents LiDAR features and B represents radar features, we are using the relationships between LiDAR and radar features to reweight the LiDAR features. The final set of output features in this case would contain LiDAR features, but their significance would be determined by the relationships found between LiDAR and radar features. This version of cross attention allows for the incorporation of radar features' information to influence the final output, while still maintaining the output to be based on the LiDAR features.

Compared to CrossAttn2, which uses the attention scores between A and A to reweight B. In the context of LiDAR and radar features, where A represents LiDAR features and B represents radar features, we are using the relationships within the LiDAR features to reweight the radar features. The final set of output features in this case would contain radar features, but their significance would be determined by the relationships found within the LiDAR features. This version of cross attention allows for the incorporation of radar features into the final output, while still being guided by the structure present in the LiDAR features.

For all previous methods then, we can utilize these variations of cross attention to determine whether there exists one variation that is best suited for fusing LiDAR and radar information

3.7 Training details

3.7.1 Baseline training

For IASSD-L, we apply the following scene-level and object level. The detailed settings are as follows:

- Random world flip along x axis.
- Random world rotation along z axis with random angle with range $[-\frac{\pi}{4}, \frac{\pi}{4}]$.
- Random world scaling with random factors with range [0.95, 1.05] Object-level augmentation
- Random objects from other scenes with more than 5 points are added to the current scene. Number of objects samples per class is as follows:
  - Car: 20
  - Pedestrian: 15
  - Cyclist: 15

We use a learning rate of 0.01 and the Adam one-cycle optimizer with a weight decay of 0.2 and momentum of 0.9 to train IASSD-L with a maximum number of epochs of 80.

3.7.2 Finetuning procedure

For all methods that involve two separate feature extraction backbones, we employ the following common finetuning procedure that is common for transfer learning tasks [45]. First, we train the baselines IASSD-L and IASSD-R from scratch, the weights from the epoch with the best validation set preformance is then used to initialize all the corresponding weights in the combined architecture. If the combined architecture has any new modules or layers, those are initialized as normal. This process is illustrated in Figure 3.20. Since there is only one prediction head in a combined model, the weights from IASSD-L are used to intialize the combined prediction head. Therefore only weights from the feature extraction backbone is used from IASSD-R.

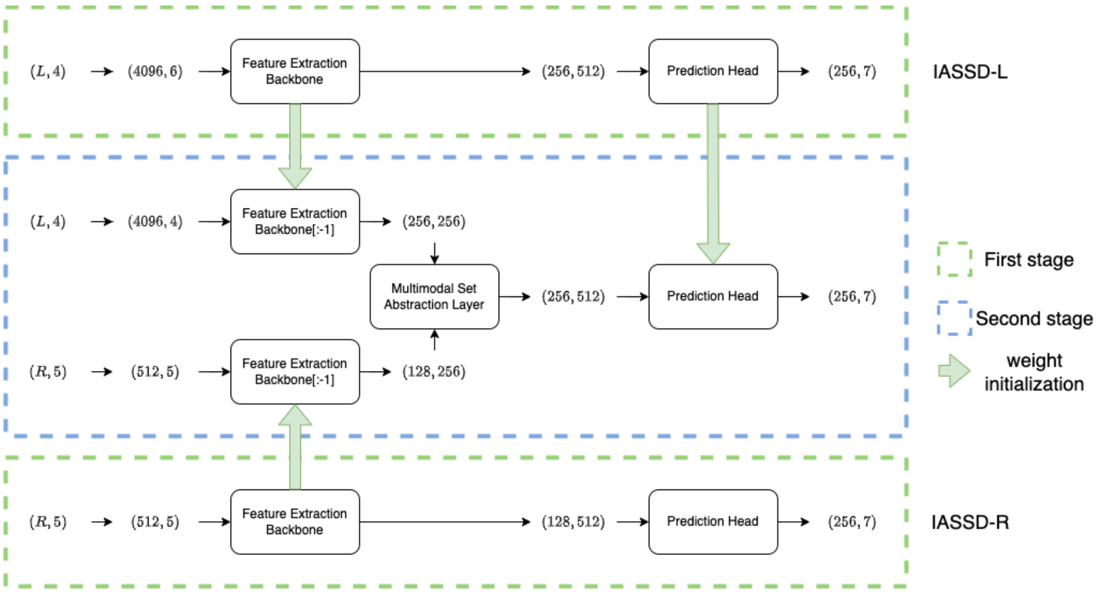


Figure 3.20: Example of the finetuning procedure. The respective backbones are training separately in the first stage, then their weights are used to intialize the corresponding weights in the combined architecture.

Chapter 4

Experimental Results

In this chapter, we present and discuss the performance results of

- Baseline methods: 3DETR, encoder only 3DETR, IASSD-L, IASSD-R
- Attention-free methods: Point cloud concatenation, feature interpolation, feature fusion with MLP
- Attentional methods: MMSA layer with single cross attention layer and cross attention block

4.1 Baseline Performance

In the following section, we present the performance results of the baseline methods, which is then used for comparison to the various other methods tested.

4.1.1 Transformer Baseline

We report the best results of adapting 3DETR in Table 4.1 on LiDAR points only. These results mostly match the findings of [36], who also discovered that 3DETR

Model	Car 3D IoU=0.5	Pedestrian 3D IoU=0.25	Cyclist 3D IoU=0.25	mAP
3DETR	did not converge			
encoder only 3DETR	17.91	0.82	2.22	6.98

Table 4.1: Performance of 3DETR and encoder only 3DETR

was largely ineffective for outdoor scenes. Figures 4.1 and 4.2 are examples of the predictions for the respective models.

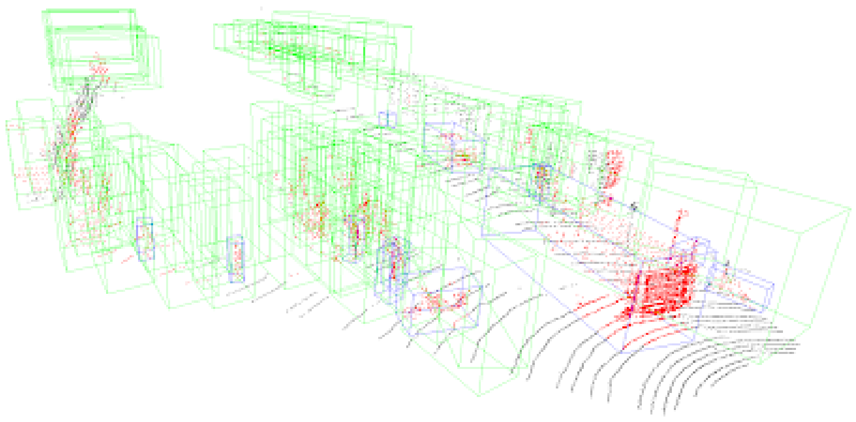


Figure 4.1: Sample output of 3DETR, ground truth bounding boxes are in blue, and predicted bounding boxes are in green.

To investigate why 3DETR fails to converge and deliver competitive benefits, we can visualise the input points, along with the sampled seed points used for the decoder. An example scene is illustrated in Figure 4.3. We observe that the seed points sampled extremely evenly throughout the scene, as expected when using FPS. Since the seed points and their features are ultimately what is used to regress into bounding boxes, this results in predicted bounding boxes being evenly

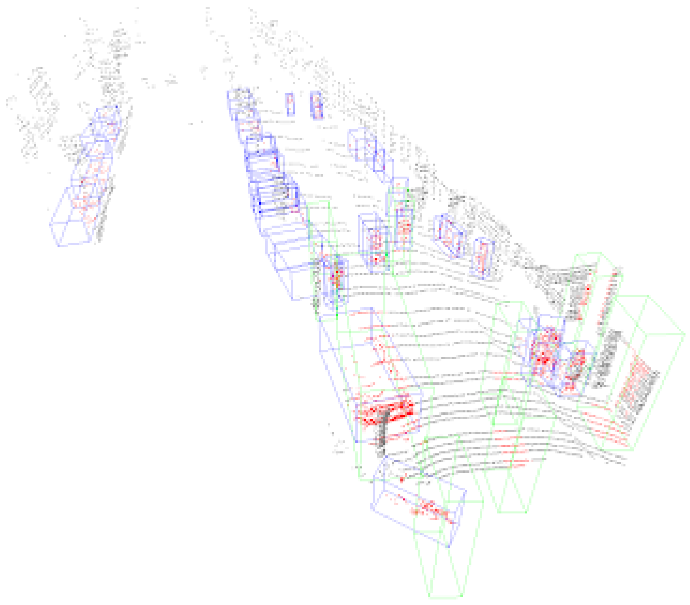


Figure 4.2: Sample output of encoder only 3DETR, ground truth bounding boxes are in blue, and predicted bounding boxes are in green.

spread across the entire scene. The visualization in Figure 4.1 supports this claim as well. Furthermore, this demonstrates the need of specialized downsampling algorithms in addition to FPS, such as F-FPS as proposed in 3DSSD [10], and center-aware sampling by IASSD [3]. Finally, this also illustrates how object detection models developed for indoor datasets are not transferrable to outdoor datasets. As shown in Figure 4.4, point cloud representing indoor scenes are much more dense compared to outdoor scenes. Therefore using FPS to select seed points is a valid method since a higher percentage of the sampled seed points would belong to some object, which is the opposite for outdoor scenes as shown in Figure 4.3.

Due to the already poor results of attempting to adapt 3DETR on LiDAR points only, the focus of the study became more focused on research questions 2 and 3: Investigating how we can build upon state of the art architectures with and without attention for outdoor scenes. The following sections report on these findings.

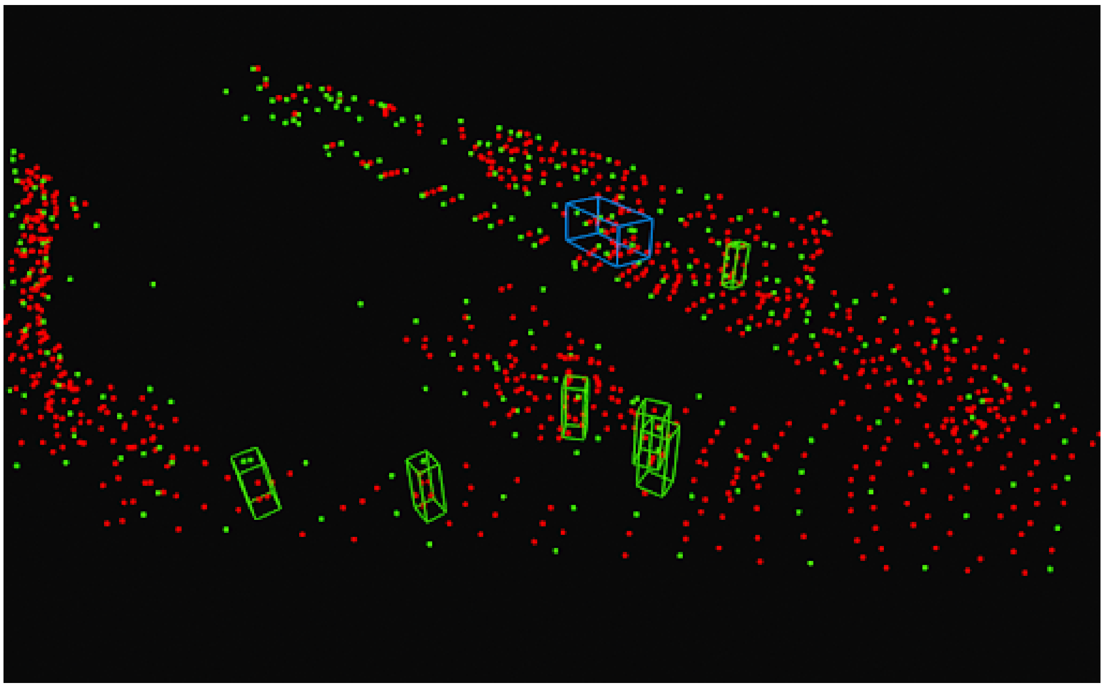


Figure 4.3: Visualization of seed points used in 3DETR. Input point cloud is in red, and sampled seed points are in green. Ground truth bounding boxes are also displayed.

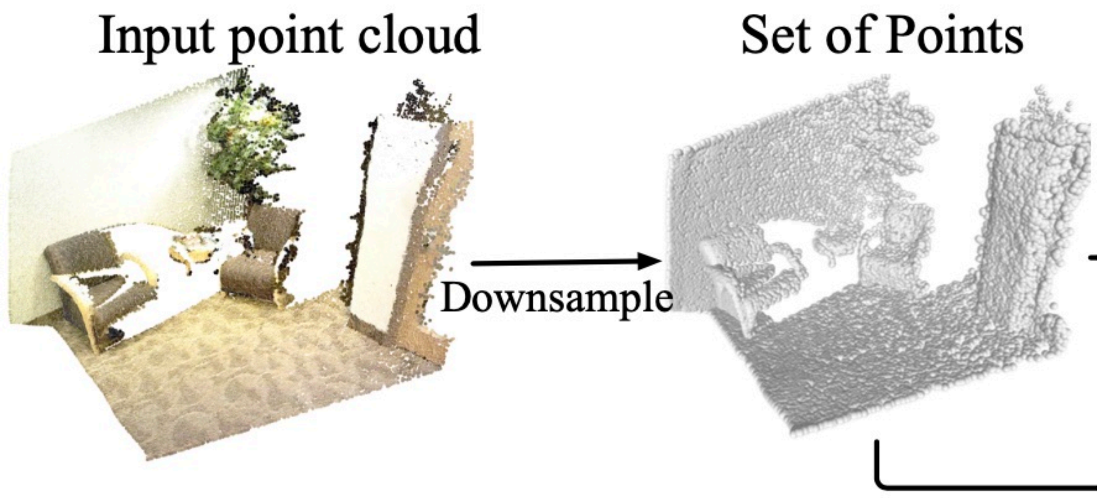


Figure 4.4: Cropped image of Figure 2.7 illustrating sample input point cloud of an indoor scene

4.1.2 IASSD Baseline

	Car 3D IoU=0.5	Pedestrian 3D IoU=0.25	Cyclist 3D IoU=0.25	mAP
IASSD-L	**78.66**	**42.56**	**67.24**	**62.82**
IASSD-R	31.24	32.50	60.69	41.48

Table 4.2: Performance of baseline modules. IASSD-L and IASSD-R are IASSD trained only on LiDAR and radar point cloud as input respectively. Best performance is **in bold**

First to establish the baseline model performance, the baseline model IASSD was trained separately on LiDAR and radar, the results are reported in Table 4.2. Unsurprisingly, we find that IASSD-L outperformed IASSD-R in every category. Given the sparsity of radar point clouds, it lacks the geometric information that LiDAR point clouds contain, thus explaining the large gap in performance for larger objects like the car class. However for the pedestrian and cyclist classes, we find that the performance gap is not as large. This may be because of the smaller size of the objects, thus geometry playing less of a role. Additionally, radar has additional information in the form of RCS and doppler velocity, thus enabling it to classify cyclist rather well given the unique feature footprint of the objects in the class. I.e. velocity of cyclists are much different than pedestrians.

Finally, this establishes the performance baselines for the attention-free and attentional methods. Given the best performing baseline (IASSD-L), one should expect a well designed architecture that takes advantage of both modalities to outperform IASSD-L in all classes.

4.2 Attention-free methods

Now that we have a baseline to compare the performance of our methods, we will now present the results of the three attention-free methods introduced earlier.

4.2.1 Point Cloud Concatenation

	Car 3D IoU=0.5	Pedestrian 3D IoU=0.25	Cyclist 3D IoU=0.25	mAP
IASSD-L	**78.66**	42.56	**67.24**	**62.82**
CONCAT	61.84 (-16.82)	**49.13** (+ 6.57)	67.06 (-0.18)	59.34 (-3.84)

Table 4.3: Performance of point cloud concatenation

Point cloud concatenation is perhaps the most straightforward method, and its performance is displayed in Table 4.3. We observe that this simplistic method does yield performance gain for the difficult pedestrian class, exhibits reduced performance for the cyclist class minimally, and shows significant decrease in performance for the car class. The resulting performance is not entirely surprising, given the nature of the concatenated point cloud. Given the difference in statistical properties of LiDAR and radar point clouds, e.g. number of points per scene, range of values for each feature, there exists two statistical distributions in the concatenated point cloud. An analogy would be similar to a mixture of gaussians as illustrated in the toy example of Figure 4.5.

Since all points share the same MLP, the MLP must learn a valid mapping for both LiDAR and radar, thus only mapping from some space that lies within the intersection between both domains to the target dimension. This is supported by the fact that the performance of IASSD-R and IASSD-L are closest in the order of cyclist, pedestrian, and car (Table 4.2) and this behaviour is exhibited in the results of the concatenation method as well. The performance disparity

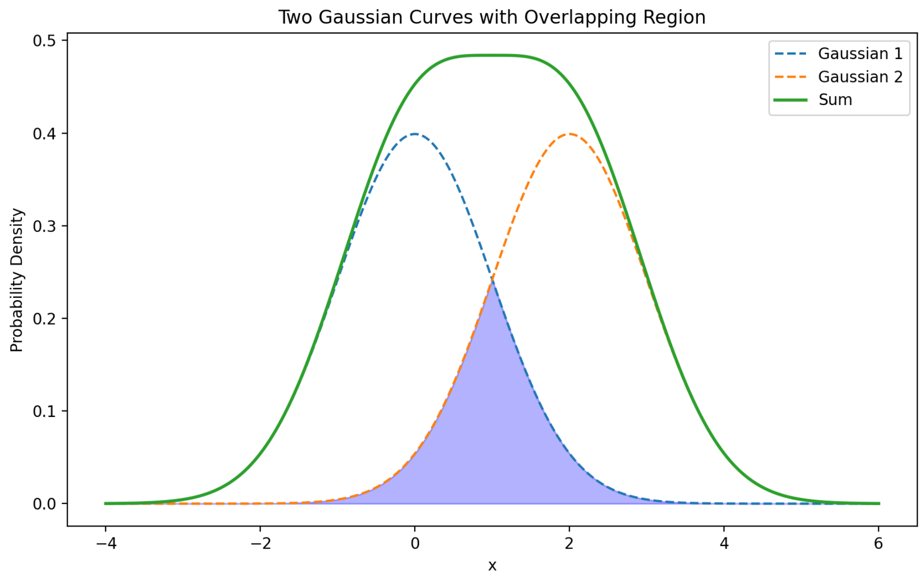


Figure 4.5: toy example of two overlapping gaussian distributions

being the largest for the car class is not surprising given the amount of geometric information provided by the numerous LiDAR points covering a car's surface. However, it is not uncommon for the same car to only have a handful of radar points. Therefore we can clearly observe how disjoint the geometric properties of LiDAR and radar point clouds are for the car class, leading to the degradation in performance when using point cloud concatenation. For the pedestrian class, the performance improvement can be attributed to the fact that pedestrians are relatively small and have less geometric detail compared to cars, the intersection between the domains may have been unique enough to sufficiently differentiate it from other classes. In the case of the cyclist class, the performance degradation is minimal, possibly because the statistical properties of LiDAR and radar point clouds for cyclists are more similar than for cars. Cyclists have more geometric detail than pedestrians but less than cars, so the concatenated point cloud can still provide useful information for the MLP to process.

In conclusion, point cloud concatenation as a method for combining LiDAR and radar data has its limitations, mainly due to the differing statistical properties of the two types of point clouds. While it does show some promise in improving performance for certain classes, like pedestrians, it may not be the most effective approach overall. Perhaps in scenarios where pedestrian and cyclists make up a majority of the objects of interest, higher false negative rates are acceptable, and the system has a very low power budget, then this may be an acceptable method. Examples of this may be autonomous mall robots that roam around and spray disinfectants [46].

4.2.2 Feature interpolation

k	r	Car 3D IoU =0.5	Pedestrian 3D IoU =0.25	Cyclist 3D IoU =0.25	mAP
1	0.1	45.78	35.27	42.32	41.12
1	0.5	58.65 (-20.01)	**54.03** (+11.47)	**68.17** (+0.93)	60.28 (-2.54)
1	1	48.73	37.68	57.24	47.88
3	0.1	44.22	36.11	51.15	43.82
3	0.5	45.17	37.55	50.67	44.46
3	1	40.14	37.88	49.28	42.43
5	0.1	45.86	36.89	50.27	44.34
5	0.5	39.89	36.76	47.50	41.38
5	1	38.8	35.98	49.94	41.92
IASSD-L		**78.66**	42.56	67.24	**62.82**

Table 4.4: Performance of feature interpolation with different number of nearest neighbours k within a search radius r meters.

The performance of feature interpolation is given in Table 4.4. Figures 4.7 and 4.8 displays the mAP plotted as a function of k and r respectively The rationale of choosing radii $r = 0.1, 0.5, 1.0$ stems from the hypothesis that we are testing with feature interpolation, that we want to approximate what a hypothetical sensor with the capabilities of both LiDAR and 4D radar would be like. Therefore the radar features interpolated onto a LiDAR point must lie within some close proximity of each other for the approximation to make sense. At larger radii, it is possible that the radar features interpolated may not even belong to the same object. For example, a radar point belonging to a car may be wrongly interpolated to a nearby pedestrian.

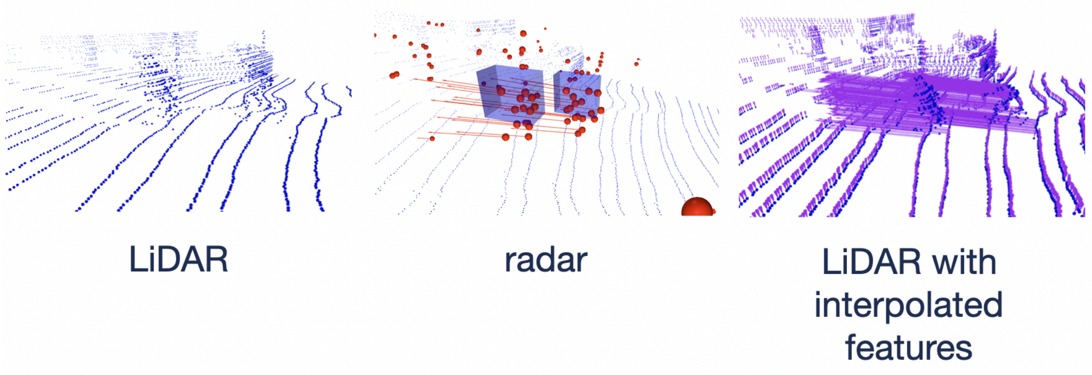


Figure 4.6: Visualization of the feature propagation module, length of the purple vector corresponds to magnitude of velocity

Interestingly, the best hyperparameter values of $k = 1$ and $r = 0.5$ outperformed the baseline for pedestrian and cyclist class with an increase of Average Precision of 11.47 and 0.93 respectively. This may illustrate that when we interpolate features, using only 1 nearest neighbor best approximates the hypothesized ideal point cloud.

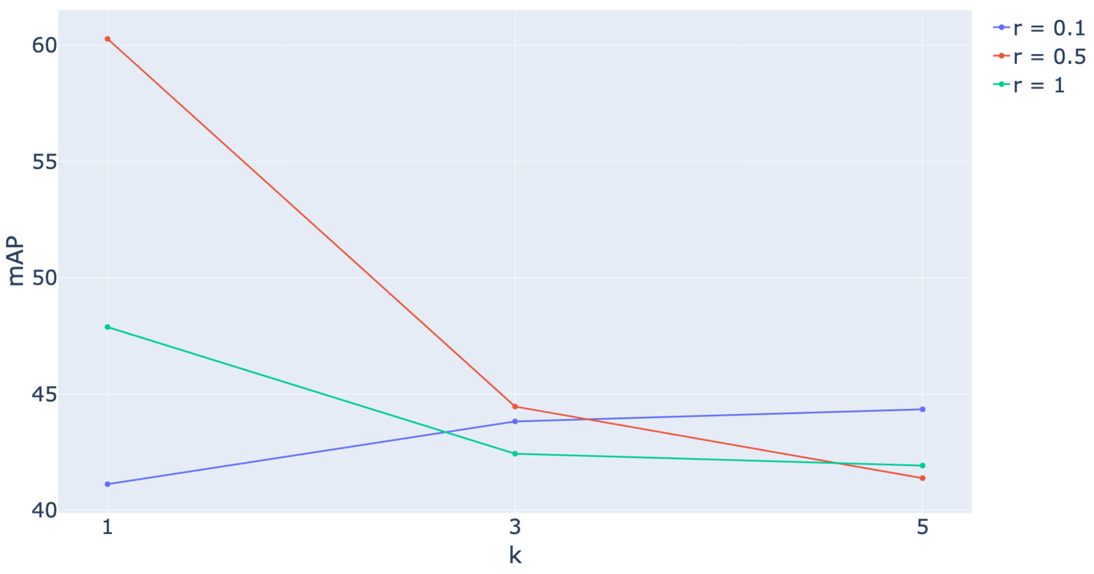


Figure 4.7: mAP vs k

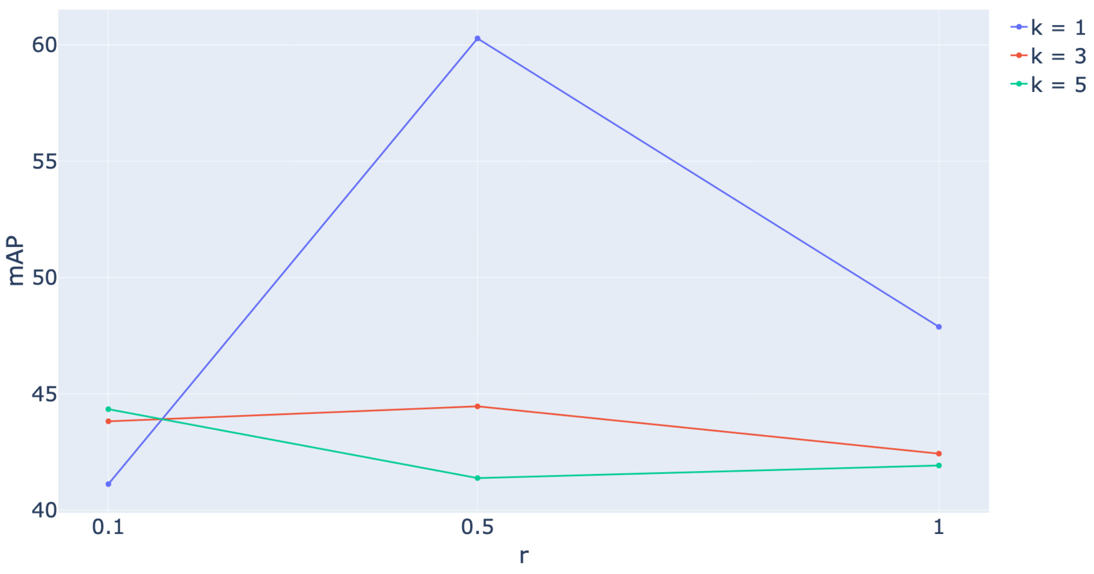


Figure 4.8: mAP vs r

4.2.3 Feature fusion with MLP

This method is relatively conceptually straightforward, since we are essentially taking the final proposal points in both modalities, matching to the nearest neighbour, and passing both sets of features through a MLP. While the computation cost does roughly double, it is straightforward to implement and yields significant performance benefits for the pedestrian and cyclist classes. Various hyperparameter settings have been tested and results displayed in Table 4.5 with AP increase of 15.00 and 5.4 respectively.

L	L_{dim}	Car 3D IoU=0.5	Pedestrian 3D IoU=0.5	Cyclist 3D IoU=0.5	mAP
1	512	71.47	**57.56**	**72.78**	**67.27**
1	256	71.79	50.27	70.81	64.29
4	512	71.29	52.27	68.81	64.13
4	256	70.82	42.30	73.30	62.14
IASSD-L		**78.66**	42.56	67.24	**62.82**

Table 4.5: MLP fusion performance. L denotes the number of layers, L_{dim} denotes the dimension of the linear layer in the MLP.

Notice however that across all hyperparameter settings, the performance of the car class worsens. The AP at a minimum dropped by 7.19. This may be an indication that the MLP, in order to fuse the features from LiDAR and radar together, guides the finetuning process in such a way that it degrades certain features from the LiDAR side, perhaps some geometric information, given the size of the car class compared to the other two classes. This would also explain why having a skip connection did not prevent the performance loss.

4.3 Attentional methods

Attention type	Query	Key	Value	Car	Pedestrian	Cyclist	mAP
single	LiDAR	radar	radar	71.56	56.08	71.20	66.28
single	LiDAR	radar	LiDAR	71.43	57.02	69.23	65.89
block	LiDAR	radar	radar	70.60	57.00	**74.57**	67.39
block	LiDAR	radar	LiDAR	**71.45**	**57.25**	73.97	**67.56**
	IASSD-L			**78.66**	42.56	67.24	62.82

Table 4.6: Attentional methods. Single: single cross attention layer, block: cross attention block as illustrated in 3.18

The performance results for all attentional methods are presented in Table 4.6. We observe that while there exist a performance increase over IASSD-L, the performance increases are remarkably similar to that result of feature fusion with a simple one layer MLP as shown in section 4.2.3. This is a rather surprising result given the theoretical ability of the attention mechanism to efficiently weight and aggregate input feature vectors by assigning higher importance to relevant features. Especially even when using an encoder style block which has more parameters compared to the architecture used when testing feature fusion with a MLP.

A closer examination of the attention weights reveals the answer. We discover that across the two sampling scales in the MMSA layer, each radar cluster on average has 13 duplicate features, with 22% of clusters having all identical features. When all the radar features are the same, the attention weights are identically $1/n$ where n is the number of features in the sampled group. In such cases, this implies that either $\text{rank}(Q) = 1$ or $\text{rank}(K) = 1$, where Q and K is the matrix of query and key feature vectors. Figure 4.9 illustrates the mathematical operations in the attention mechanism more clearly. Each row in the attention matrix is the

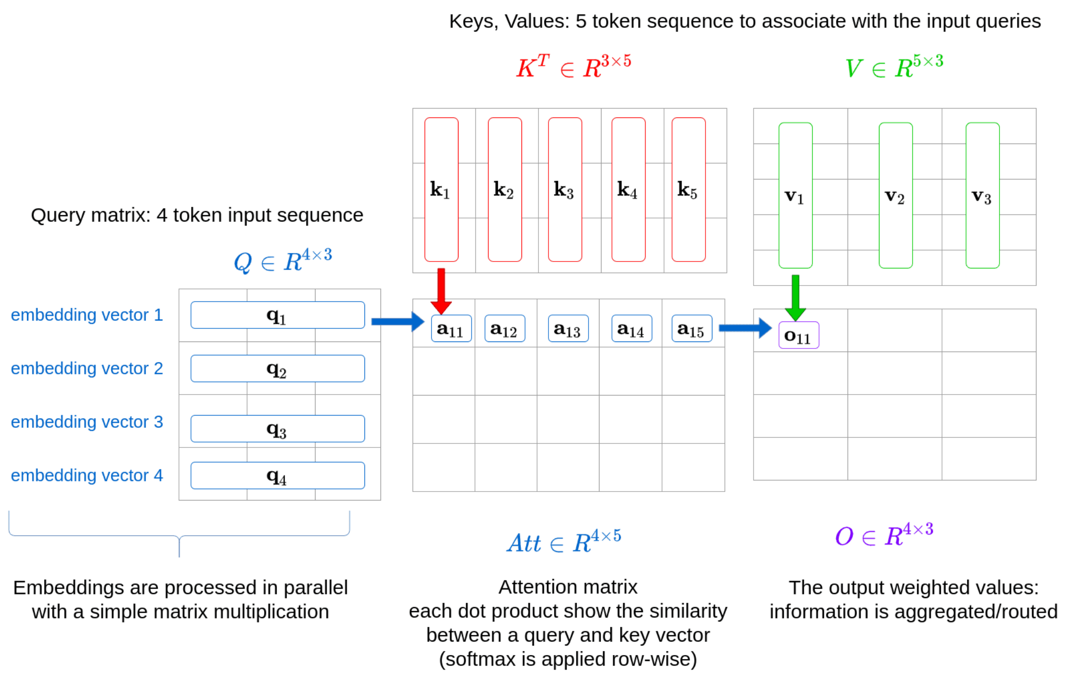

Keys, Values: 5 token sequence to associate with the input queries
$K^T \in R^{3\times5}$
$V \in R^{5\times3}$
Query matrix: 4 token input sequence
$Q \in R^{4\times3}$
embedding vector 1
embedding vector 2
embedding vector 3
embedding vector 4
$\mathbf{q}_1$
$\mathbf{q}_2$
$\mathbf{q}_3$
$\mathbf{q}_4$
$\mathbf{k}_1$ $\mathbf{k}_2$ $\mathbf{k}_3$ $\mathbf{k}_4$ $\mathbf{k}_5$
$\mathbf{v}_1$ $\mathbf{v}_2$ $\mathbf{v}_3$
$\mathbf{a}_{11}$ $\mathbf{a}_{12}$ $\mathbf{a}_{13}$ $\mathbf{a}_{14}$ $\mathbf{a}_{15}$
$\mathbf{o}_{11}$
$Att \in R^{4\times5}$
$O \in R^{4\times3}$
Embeddings are processed in parallel with a simple matrix multiplication
Attention matrix each dot product show the similarity between a query and key vector (softmax is applied row-wise)
The output weighted values: information is aggregated/routed

Figure 4.9: Another illustration of the attention mechanism [7]

dot product of a query and key vector, or that

$$a_{ij} = q_i \cdot k_j$$

once the attention matrix is calculated, the softmax operator is applied row-wise. i.e.

$$a_{ij} = \frac{\exp(q_i \cdot k_j)}{\sum_{m=1}^{n} \exp(q_i k_m)}$$

these scores are then used as the weights to compute the weighted sum of the value vectors as illustrated in Figure 4.9. However, observe that if $k_1 = k_2 = \cdots = k_n$, the attention weights become

$$\begin{aligned} a_{ij} &= \frac{\exp(q_i \cdot k_1)}{\sum_{m=1}^{n} \exp(q_i k_1)} \\ &= \frac{\exp(q_i \cdot k_1)}{n \cdot \exp(q_i \cdot k_1)} \\ &= \frac{1}{n} \end{aligned}$$

which means that the every single output feature vector will just be an average of the value vectors. In cross attention, recall that the keys and value vectors are the same. Therefore the output feature vectors will just be identical to the keys. Thus we have inadvertently shown the following, albeit somewhat trivial, theorem:

Theorem 1 *(Attention is Identity) Applying the attention mechanism*

$$S = \operatorname{soft\,max}\left(\frac{QK^T}{\sqrt{d_q}}\right)V$$

with $Q \in \mathbb{R}^{d \times d_q}$, $K \in \mathbb{R}^{d \times d_k}$, $V \in \mathbb{R}^{d \times d_v}$, *is identical to*

$$S = \mathbf{I}V$$

with $\mathbf{I}$ *being the identity matrix, if either* $rank(Q) = 1$ *or* $rank(K) = 1$

Therefore, the output features from applying cross attention between LiDAR and radar features are simply the radar features, since a skip connection is applied afterwards anyway. Applying cross attention has no effect if all the radar features are the same. The overall result is approximately the same as feature fusion with MLP, as the linear layers, normalization, and dropout layers in the cross attention modules affect the performance in a negligible manner.

But how frequent is it that we have radar clusters with identical features? We take the sampled groups from the MMSA layer at both sampling scales of $r = 4.8$ and $r = 6.4$ and check the number of points with identical features across the entire validation set. The results are displayed in Figure 4.10, where *no ckpt* represents the model at the beginning of training, i.e. backbone weights are identical to IASSD-L and IASSD-R, and '*with ckpt*' represents the model at the end of training in the graph.

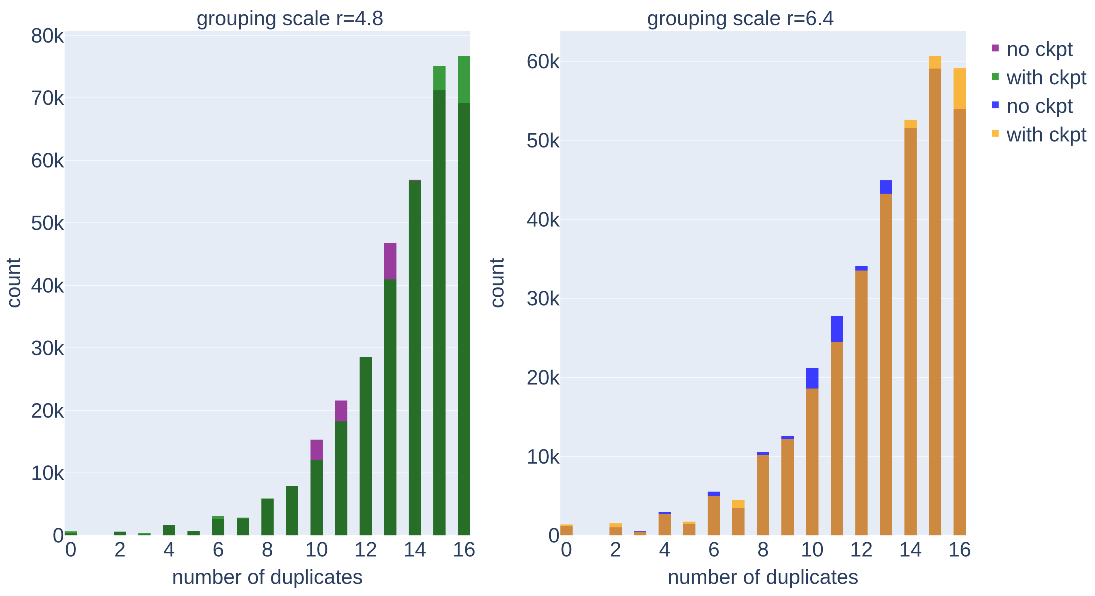


Figure 4.10: Number of duplicates in radar point clusters in MMSA layer for each sampling range

What we observe is that in both cases and at both scales, a majority of grouped radar points actually have duplicate features. Table 4.7 illustrates the average number of duplicated features per radar cluster.

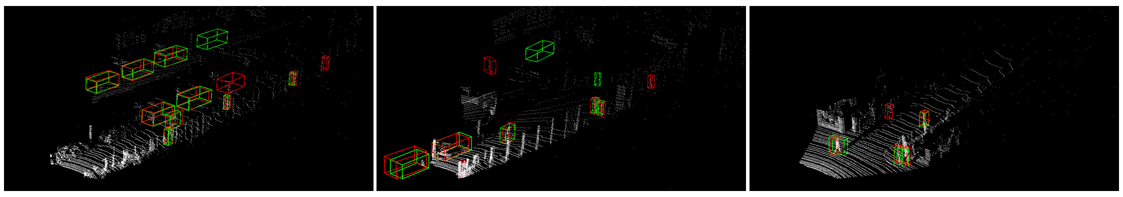
scale	ckpt	average duplicates
$r = 4.8$	✓	13.60
$r = 4.8$	X	13.49
$r = 6.4$	✓	12.92
$r = 6.4$	X	12.84

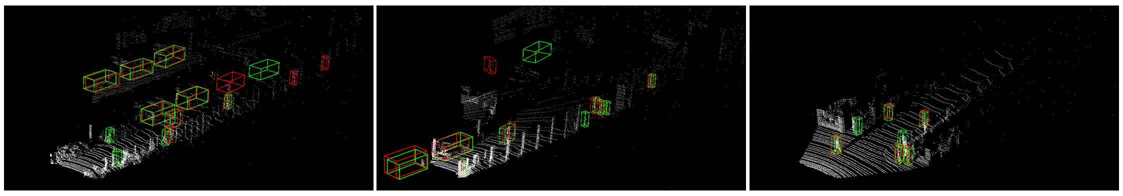
Table 4.7: Average number of points with duplicate features for each sampling scale, where 16 points are sampled.

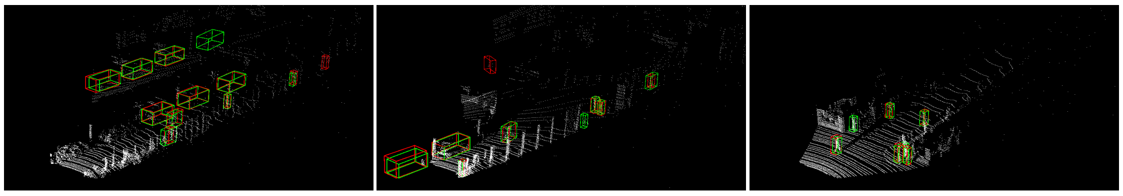
Therefore in practice, most instances of applying cross attention between a group of LiDAR and radar points is still approximately doing nothing, since for clusters with a high number of the duplicates, the remaining unique features will contribute very little to the output features. Furthermore, since the set of output features are passed through a maxpool operator, the dominant duplicated features will likely be selected anyway. Therefore explaining why all attentional methods are similar in performance with feature fusion with MLP.

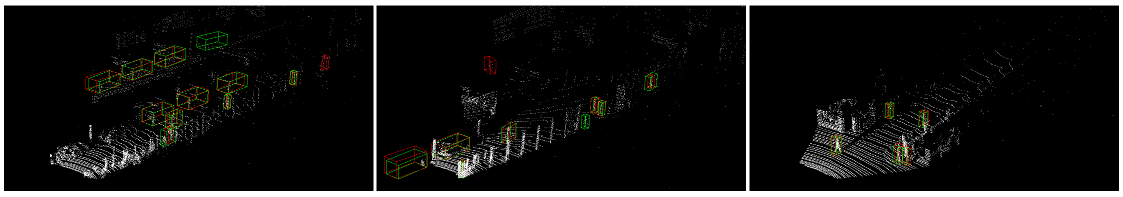
4.4 Qualitative Results

Qualitative results for each method are displayed in Figure 4.11, best viewed in color and zoomed in for detail.


(a) Point Cloud Concatenation


(b) Feature Interpolation


(c) Feature fusion with MLP


(d) MMSA layer with single cross attention layer

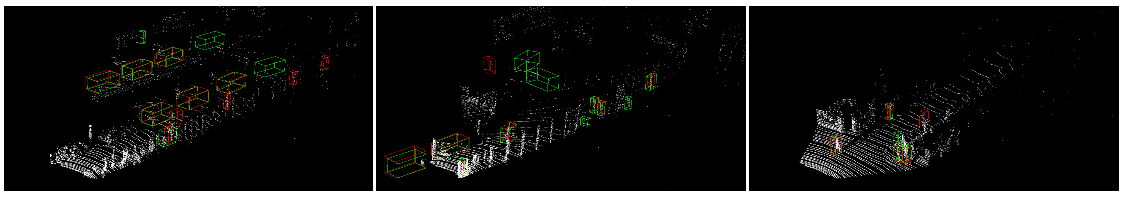

(e) IASSD-L

Figure 4.11: Qualitative results for different methods. Red bounding box indicates the ground truth, while green bounding box represents model predictions.

Chapter 5

Conclusions

In this thesis we have proposed several attentional and attention free methods to fuse information from LiDAR and 4D radar point clouds in order to increase performance. We first discovered that an encoder-decoder style network like 3DETR that was designed for indoor datasets was unable to be adopted for outdoor datasets like VoD. Attention free methods included point cloud concatenation, feature propagation and feature fusion with MLP. Out of the three, feature fusion with MLP was the only method to outperform IASSD in overall performance. Furthermore, several attentional methods were tested through the introduction of the multimodal set abstraction layer, which attempted to combine information from both modalities by utilizing the cross attention mechanism. However, due to how radar points within a sampled neighbourhood have duplicated features, the cross attention mechanism was reduced to approximately applying the identity matrix to the set of radar features. As a result, the attentional methods perform very similarly to the feature fusion with MLP.

Our findings point to possible future research directions. Firstly, while IASSD-R appears to perform best with the current downsampling point settings, this results in the feature duplication phenomenon as described earlier. While the duplication does not affect single modality (radar) performance, one avenue of research could be to further investigate to what extent this phenomenon improves or hinders the model's performance. Secondly, given that the point features are more diverse after the first set abstraction layer, perhaps a more optimal way to fuse features from both modalities would be in the earlier stages of the feature extraction backbone.

References

[1] A. Palffy, E. Pool, S. Baratam, J. F. P. Kooij, and D. M. Gavrila, "Multi-class road user detection with 3+1d radar in the view-of-delft dataset," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4961–4968, 2022. ii, 30, 38

[2] I. Misra, R. Girdhar, and A. Joulin, "An End-to-End Transformer Model for 3D Object Detection," in *ICCV*, 2021. ii, 16, 27

[3] Y. Zhang, Q. Hu, G. Xu, Y. Ma, J. Wan, and Y. Guo, "Not all points are equal: Learning highly efficient point-based detectors for 3d lidar point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18953–18962, 2022. ii, vii, 18, 24, 35, 59

[4] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *CVPR*, 2020. vi, 3, 6, 30

[5] R. Nabati and H. Qi, "Radar-camera sensor fusion for joint object detection and distance estimation in autonomous vehicles," *arXiv preprint arXiv:2009.08428*, 2020. vi, 6

[6] C. Urmson, J. A. Bagnell, C. Baker, M. Hebert, A. Kelly, R. Rajkumar, P. E. Rybski, S. Scherer, R. Simmons, S. Singh, *et al.*, "Tartan racing: A multi-modal approach to the darpa urban challenge," 2007. vi, 3, 7, 8, 9

[7] T. A. S. Team, “Why multi-head self attention works: math, intuitions and 10+1 hidden insights.” Accessed: 2023-04-22, 2021. Available online at: `https://theaisummer.com/self-attention/`. vi, ix, 13, 14, 70

[8] C. R. Qi, O. Litany, K. He, and L. J. Guibas, “Deep hough voting for 3d object detection in point clouds,” in *proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9277–9286, 2019. vii, 20

[9] S. Shi, X. Wang, and H. Li, “Pointrcnn: 3d object proposal generation and detection from point cloud,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 770–779, 2019. vii, 21, 22, 35

[10] Z. Yang, Y. Sun, S. Liu, and J. Jia, “3dssd: Point-based 3d single stage object detector,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11040–11048, 2020. vii, 18, 23, 35, 59

[11] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, “Pct: Point cloud transformer,” *Computational Visual Media*, vol. 7, pp. 187–199, 2021. vii, 25, 26

[12] X. Pan, Z. Xia, S. Song, L. E. Li, and G. Huang, “3d object detection with pointformer,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7463–7472, June 2021. vii, 28, 51

[13] M. Takahashi, Y. Ji, K. Umeda, and A. Moro, “Expandable yolo: 3d object detection from rgb-d images,” in *2020 21st International Conference on Research and Education in Mechatronics (REM)*, pp. 1–5, IEEE, 2020. vii, 36

[14] KryoTech, “Object Detection for Self-Driving Cars.” `https://kryotech.co.`

uk/object-detection-for-self-driving-cars/, 2021. Accessed: 2023-04-22. vii, 36

[15] J. Hui, "MAP: Mean Average Precision for Object Detection." https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173, 2018. Accessed: 2023-04-22. viii, 38

[16] A. LaFrance, "Your grandmother's driverless car," Jun 2016. 2

[17] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," *Advances in neural information processing systems*, vol. 1, 1988. 2

[18] R. Behringer, S. Sundareswaran, B. Gregory, R. Elsley, B. Addison, W. Guthmiller, R. Daily, and D. Bevly, "The darpa grand challenge-development of an autonomous vehicle," in *IEEE Intelligent Vehicles Symposium, 2004*, pp. 226–231, IEEE, 2004. 2

[19] S. Thrun, "Winning the darpa grand challenge," in *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*, pp. 4–4, Springer, 2006. 3

[20] C. M. University, "Uber, carnegie mellon announce strategic partnership and creation of advanced technologies center in pittsburgh - news - carnegie mellon university." 3

[21] K. Korosec, "Elon musk says tesla vehicles will drive themselves in two years," Apr 2021. 3

[22] "Waymo launches nation's first commercial self-driving taxi service in arizona," Dec 2018. 3

[23] P. Dong and Q. Chen, *LiDAR remote sensing and applications.* CRC Press, 2017. 4

[24] H. Laga, L. V. Jospin, F. Boussaid, and M. Bennamoun, “A survey on deep learning techniques for stereo-based depth estimation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 4, pp. 1738–1764, 2020. 4

[25] A. J. Hawkins, “Mercedes-benz and luminar expand their partnership on lidar and adas,” *The Verge*, Feb 2023. 5

[26] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017. 8, 9

[27] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” *Advances in neural information processing systems*, vol. 30, 2017. 8, 10, 19

[28] R. B. Rusu, N. Blodow, and M. Beetz, “Fast point feature histograms (fpfh) for 3d registration,” in *2009 IEEE international conference on robotics and automation*, pp. 3212–3217, IEEE, 2009. 9

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017. 11, 27, 48, 49

[30] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, “Frustum pointnets for 3d object detection from rgb-d data,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 918–927, 2018. 19

[31] J. Illingworth and J. Kittler, "A survey of the hough transform," *Computer vision, graphics, and image processing*, vol. 44, no. 1, pp. 87–116, 1988. 20

[32] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pp. 213–229, Springer, 2020. 27

[33] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5828–5839, 2017. 27

[34] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 567–576, 2015. 27

[35] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*, pp. 3354–3361, IEEE, 2012. 27, 28, 30

[36] C. He, R. Li, S. Li, and L. Zhang, "Voxel set transformer: A set-to-set approach to 3d object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8417–8427, 2022. 27, 57

[37] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF*

Conference on Computer Vision and Pattern Recognition (CVPR), June 2020. 30

[38] L. Zheng, Z. Ma, X. Zhu, B. Tan, S. Li, K. Long, W. Sun, S. Chen, L. Zhang, M. Wan, *et al.*, "Tj4dradset: A 4d radar dataset for autonomous driving," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 493–498, IEEE, 2022. 31

[39] D.-H. Paek, S.-H. Kong, and K. T. Wijaya, "K-radar: 4d radar object detection for autonomous driving in various weather conditions," in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. 31

[40] O. D. Team, "Openpcdet: An open-source toolbox for 3d object detection from point clouds." `https://github.com/open-mmlab/OpenPCDet`, 2020. 31

[41] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12697–12705, 2019. 32

[42] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014. 47

[43] P. Li, J. Gu, J. Kuen, V. I. Morariu, H. Zhao, R. Jain, V. Manjunatha, and H. Liu, "Selfdoc: Self-supervised document representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5652–5660, 2021. 53

[44] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *Advances in neural information processing systems*, vol. 28, 2015. 53

[45] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, “Learning and transferring mid-level image representations using convolutional neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1717–1724, 2014. 56

[46] “Robomop? pandemic boosts demand for cleaning robots in hong kong malls,” Apr 2021. 65

[47] B. Rimé and L. Schiaratura, “Gesture and speech in fundamentals of nonverbal behavior,” 01 1991.

[48] E. Coronado, J. Villalobos, B. Bruno, and F. Mastrogiovanni, “Gesture-based Robot Control: Design Challenges and Evaluation with Humans,” 05 2017.

[49] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 922–928, IEEE, 2015.

[50] B. Li, “3d fully convolutional network for vehicle detection in point cloud,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1513–1518, IEEE, 2017.

[51] G. Pang and U. Neumann, “3d point cloud object detection with multi-view convolutional neural network,” in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 585–590, IEEE, 2016.

[52] B. Leibe, A. Leonardis, and B. Schiele, “Combined object categorization and segmentation with an implicit shape model,” in *Workshop on statistical learning in computer vision, ECCV*, vol. 2, p. 7, 2004.

[53] T. Yin, X. Zhou, and P. Krahenbuhl, “Center-based 3d object detection and tracking,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11784–11793, 2021.

[54] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, “Pv-rcnn: Point-voxel feature set abstraction for 3d object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10529–10538, 2020.

[55] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[56] X. Pan, Z. Xia, S. Song, L. E. Li, and G. Huang, “3d object detection with pointformer,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7463–7472, 2021.