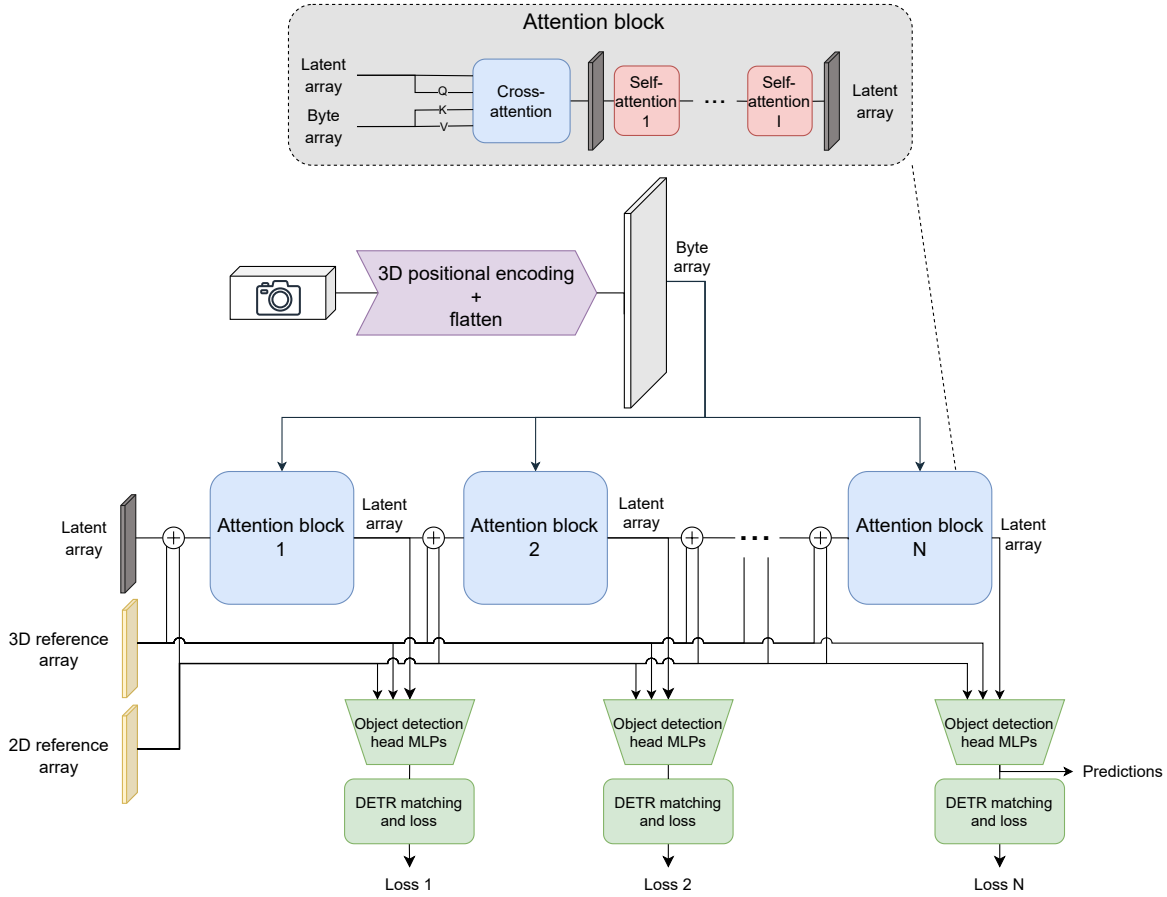
CHALMERS
UNIVERSITY OF TECHNOLOGY



3D Object Detection with Perceiver

Adapting the Perceiver Architecture for 3D Object Detection with Automotive Image Data

Master's thesis in Systems, Control and Mechatronics

Lydia Andersson and Linnéa Sedin

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

3D Object Detection with Perceiver

Adapting the Perceiver Architecture for 3D Object Detection with Automotive Image Data

Lydia Andersson and Linnéa Sedin

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

3D Object Detection with Perceiver
Adapting the Perceiver Architecture for 3D Object Detection with Automotive Image Data
Lydia Andersson and Linnéa Sedin

Supervisors: Willem Verbeke, Zenseact AB
Maryam Fatemi, Zenseact AB
Joakim Johnander, Zenseact AB
Examiner: Fredrik Kahl, Department of Electrical Engineering,
Chalmers University of Technology

Master's Thesis 2024
Department of Electrical Engineering,
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An overview of the Per3D-ObDet architecture. The last attention block is shown in detail.

Gothenburg, Sweden 2024

3D Object Detection with Perceiver
Adapting the Perceiver Architecture for 3D Object Detection with Automotive Image Data
Lydia Andersson and Linnéa Sedin
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In the field of autonomous vehicles, accurately understanding and interpreting the environment and surroundings is vital for safe navigation. A robust perception system is fundamental in achieving this, and central to this is the use of computer vision techniques. This project investigates the adaptation of Perceiver [13] for 3D object detection using automotive image data. While the Perceiver architecture, which is based on Transformers, is designed for perception tasks, it has neither been evaluated on large input sizes nor tested on 3D object detection tasks. By using the ZOD dataset [2], which provides a comprehensive collection of automotive image data, this study aims to assess Perceiver's capabilities in this context. Specifically, 3D object detection is exclusively done on camera images, targeting only vehicle objects. The proposed model in this project implements end-to-end object detection where the loss computation is inspired by the DETR loss computation [4].

The results indicate that the Perceiver architecture does not work without additional configuration for 3D object detection. Challenges in adapting the model highlight the need for further modifications and optimizations to improve its performance in this specific application. Despite these challenges, the study provides valuable insights into the potential and limitations of using the Perceiver architecture for 3D object detection with large input data. This study also provides insight into the impact of different additional modifications to the basic architecture.

Keywords: 3D Object Detection, Artificial Intelligence, Automotive Sensing, Deep Machine Learning, DETR, Perceiver, Transformer

Acknowledgements

We would like to express our deepest gratitude to our supervisors at Zenseact, Willem Verbeke, Maryam Fatemi, and Joakim Johnander, for their invaluable help and insightful guidance. This project would not have been possible without the generous support. A special thanks to Willem Verbeke for his exceptional support and contribution to this project. We are also thankful to our academic supervisor at Chalmers University of Technology, Fredrik Kahl, for his encouragement and support.

Lydia Andersson, Linnéa Sedin, Gothenburg, May 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AD	Autonomous Driving
Adam	Adaptive Moment Estimation
ADAS	Advanced Driver Assistance Systems
BEV	Birds Eye View
CNN	Convolutional Neural Network
DETR	End-to-End Object Detection with Transformers
DLA	Deep Layer Aggregation
FNN	Feed Forward Neural Network
FCN	Fully Connected Neural Network
GPU	Graphics Processing Unit
IDA	Iterative Deep Layer Aggregation
LiDAR	Light Detection and Ranging
mAP	Mean Average Precision
MLP	Multi-layer Perceptrons
NLP	Natural Language Processing
ROI	Region Of Interest
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
VRAM	Video Random-Access Memory
ZOD	Zenseact Open Dataset

Contents

1
Introduction

This chapter introduces the background of this research project. It covers the motivation and research questions, along with a discussion about the ethical, social, and sustainability aspects.

1.1 Motivation

In the domain of autonomous vehicles, the development of a robust perception system is a central focus. A crucial component in this pursuit involves utilizing computer vision for 3D object detection. The ability of self-driving vehicles to comprehend and interpret the surroundings is fundamental to ensuring safe and smooth navigation through dynamic and complex environments [10].

However, the pursuit of effective 3D object detection in autonomous vehicles faces inherent challenges. There are currently many different approaches to 3D object detection. Some techniques use only one sensor, such as cameras, while others use a multi-modal approach [21]. Traditional methods struggle with the efficient fusion of information from various sensors, including LiDAR and cameras. Furthermore, scalability, computational efficiency, and data storage are key challenges, given that 3D object detection systems must efficiently handle diverse scenarios while balancing accuracy with real-time processing demands. Addressing these challenges requires solutions capable of effectively fusing data from multiple sensors and processing information [10].

The novel neural network architecture, called Perceiver, is specifically designed to handle input from arbitrary sensor modalities, and scales efficiently with the size of the inputs. By mapping the byte array to a latent array, Perceiver achieves linear scaling with the input data size. This makes it a potentially promising architecture for enhancing 3D object detection in the context of autonomous vehicles. In the original paper, Perceiver has not been tested on object detection, nor has it been evaluated with large input sizes, which are commonly encountered in autonomous vehicles. In addition, the paper lacked a detailed study of fusion between multiple modalities [13]. Therefore, this project investigates the performance of Perceiver in 3D object detection and discusses the potential of sensor fusion.

1.2 Research Questions

The overarching purpose of this project is to investigate the process of 3D object detection through the utilization of Perceiver. The goal is to explore if and how the Perceiver architecture can be used to perform 3D object detection using automotive image data. The following research questions will be used to guide this thesis project:

- Can the Perceiver architecture be modified for 3D object detection, and if feasible, what modifications are necessary to achieve this?
- How can the training of the final model be adapted and optimized to effectively handle large input sizes, that are often encountered in the automotive industry?
- In the context of automotive data, could Perceiver facilitate sensor fusion and temporal fusion for improved 3D object detection performance?

1.3 Limitations

Some limitations have been made to narrow the scope. The project focuses specifically on automotive image data sourced from the ZOD dataset [2]. This dataset provides a comprehensive collection of automotive images, which will serve as the data source for training and evaluating the model. Furthermore, the model will only utilize camera frames for training without incorporating other sensors or sequences. The training of the model will be specifically aimed at perceiving vehicles within 100 meter distance, with no other objects being identified.

Additionally, this project focuses exclusively on exploring the Perceiver architecture and end-to-end object detection with a loss function inspired by DETR [4] for perception tasks without using any other models for perception.

1.4 Ethical and Social Aspects

In the field of artificial intelligence and the use of technologies like 3D object detection, it is crucial to be highly mindful of ethical considerations. This is particularly important when examining the diverse applications of these technologies, ranging from autonomous vehicles to sensitive areas like military applications. In terms of the automotive industry, issues related to decision-making algorithms, accountability in accidents, and the overall safety of self-driving cars require careful ethical attention. On the other hand, 3D object detection technologies have great potential in the development across diverse domains. In the automotive sector, it has the potential to prevent injuries and save lives.

In this project, the public dataset ZOD [2] will be used which has inherent limitations. Since the dataset has been collected in certain geographical locations, it inherently reflects the demographic composition and traffic scenarios of those locations. Despite these limitations, it is expected that the findings can be reasonably extended with some efforts to diverse geographical locations featuring distinct populations and traffic scenarios. Consequently, the perceived risk of bias and unfairness arising from this research will be minimal.

Nevertheless, with the development and improvements of AI technologies, unintended consequences must be prevented. Responsible governance, international collaboration, and the establishment of ethical guidelines are crucial steps to ensure the appropriate and beneficial use of AI technologies across various domains. It is essential to ensure responsible use and uphold ethical principles such as accountability and fairness.

1.5 Sustainability Aspects

United Nations has defined 17 sustainability goals [1], each relating to different parts of sustainability. In today's society, where the development of technology is evolving fast, it is important to not forget about the sustainability aspects and innovate responsibly. One of the sustainability goals considers the well-being and health of people. A key motivation behind the development of autonomous vehicles is to decrease the number of accidents and fatalities in traffic, aligning with the vision of better health.

While there are positive aspects, it is also essential to recognize the harmful impact that the automotive industry has on the climate. The existing vehicles on the roads have significantly contributed to carbon dioxide emissions, adversely affecting the climate. However, there is a potential that autonomous vehicles might drive in a more environmentally friendly way, mitigating these harmful effects and easing the strain on our climate. Furthermore, the autonomous advancement of vehicles could also have positive effects on the efficiency of traffic, especially in countries where queuing is a prevalent issue, often resulting in cars standing idle.

Additionally, the impact on the climate extends beyond the vehicles themselves. It is also noteworthy to consider the sustainability implications of large compute and data center facilities required for training deep learning models, including those used in autonomous vehicle development. These facilities consume substantial amounts of energy, contributing to the carbon footprint of AI technologies. While this aspect is often overlooked, there are ways to mitigate these environmental impacts, such as exploring more energy-efficient hardware, adopting renewable energy sources, and implementing optimization techniques for model training.

In conclusion, the evolution of autonomous vehicles yields a complex interplay of both adverse and beneficial effects on sustainability, as outlined in the 17 sustainability goals. While certain aspects contribute positively, it is essential to recognize that the industry's impact extends to various goals, with both constructive and potentially detrimental outcomes.

2
Theory

This chapter explores the fundamental concepts and theoretical frameworks underpinning the study of 3D object detection using a transformer-based neural network architecture.

2.1 Deep Machine Learning

Machine learning is a subfield of artificial intelligence where a model learns relationships and patterns in a dataset. Machine learning models can be trained for specific tasks, where some common applications are image classification, object detection, and machine translation. Neural networks are machine learning models inspired by the structure and function of the human brain [14].

There are different types of learning, and this project operates within the framework of supervised learning. In supervised learning, the model is provided with input-output pairs during the training process, commonly referred to as input and target values. The model learns by iteratively comparing its predictions to the target values provided in the dataset and adjusting its parameters to minimize the difference [8].

The learning process unfolds in three key parts: forward pass, loss calculation, and backward propagation. In the forward pass, the input data is passed through the neural network, generating predictions. The input refers to the data fed into the network, depending on the specific task, and is converted into numbers. Following the forward pass, the loss is computed, which quantifies the offsets of the predictions and the target values. The loss then serves as a guide during the backward propagation, where adjustments to the network weights and biases are made to force it to make better predictions.

The process of forward pass, loss computation, and backward propagation is repeated multiple times for the same training dataset, called epochs. Additionally, the data can be divided into batches, where each batch is a subset of the training data. The batch size defines the number of data samples that are being passed through the model before the backward propagation.

As models progress in complexity, becoming more advanced and consisting of more layers and parameters, they transition into the realm of deep neural networks. This section introduces the fundamentals of Deep Machine Learning, establishing a foundation of knowledge necessary for comprehending the other concepts discussed in this chapter.

2.1.1 Basic Architecture of Neural Networks

Within neural networks, nodes, also known as neurons or units, are the fundamental computational units responsible for performing a computation on an input and generating an output. The nodes are arranged into layers, which can be categorized as input, hidden, and output layers. The input to the input layer is the raw data representing one sample

from the dataset. Figure 2.1 provides a clarification of the structure of a basic neural network.

The computation carried out by the node varies depending on the desired architecture of the neural network. However, the most common operation is a linear function, where each input is multiplied by a weight, to which a bias term is then added. The weights and biases are learnable parameters of the network, meaning that they are adjusted throughout the training procedure to generate the desired output of the model.

Following the linear transformation, a node applies an activation function before generating the output. The total operation performed by a node can thereby be defined as

$$y = f(\sum_i W_i x_i + b) \tag{2.1}$$

where y is the output, x_i is the i-th element of input vector x, and f is the activation function. In this equation, W represents the weight assigned to each input, and b is the bias, which is the same for all inputs to the node.

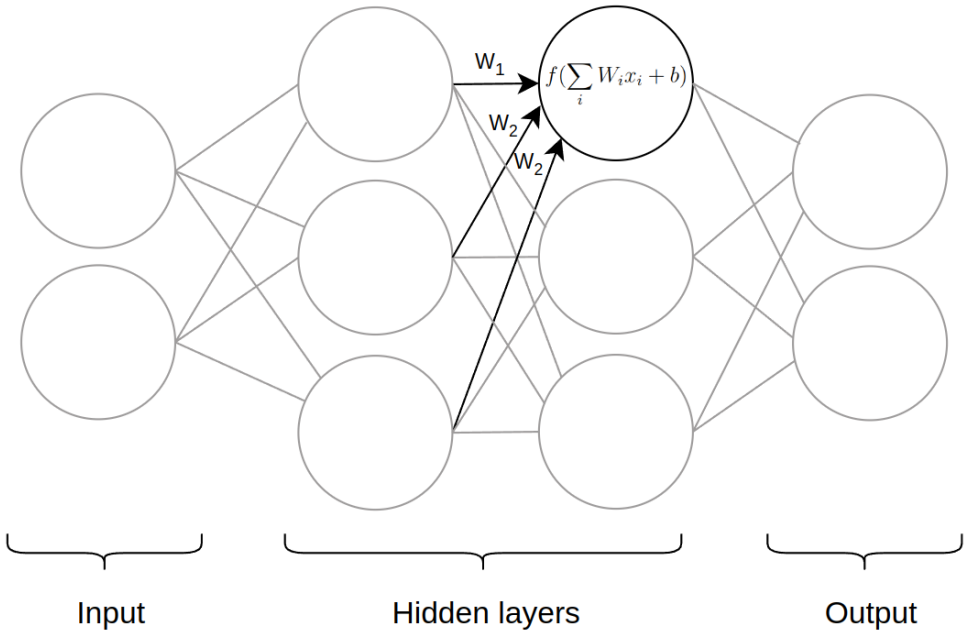


Figure 2.1: A compact and shallow neural network to visualize the computations carried out within the nodes and how nodes are arranged into layers. The operation in the highlighted node corresponds to Equation 2.1. The highlighted arrows coming from the previous layer have weights Wi assigned to them.

The activation function plays a crucial role in the neural network. Common activation functions include the rectified linear unit (ReLU) and the sigmoid function. ReLU function is defined as

$$f(x) = \max(0, x) \ . \tag{2.2}$$

It introduces non-linearity into the calculations, allowing the model to capture complex relationships and patterns, and is often used in hidden layers. Sigmoid, however, maps the input to a value between zero and one, making it suitable for binary classification tasks, among others. Its mathematical expression is

$$f(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} \ . \tag{2.3}$$

Softmax is another widely used function in machine learning and is usually applied to the output layer of neural networks. It takes a vector as input and normalizes the values so that they sum up to one, while strongly reinforcing the highest values compared to the rest of the values. This makes Softmax suitable for generating a probability distribution in multi-class classification scenarios, where the output will be a vector of probabilities for each class. Softmax is defined as

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \ . \tag{2.4}$$

The outputs from the nodes in the output layer are transformed into an input vector $\vec{x}$, where x_i refers to an element in the input vector. The summation is performed over all elements, x_j, in the input vector.

A machine learning model's performance depends heavily on the parameters within the network. While some parameters are learned during the training process, others are chosen before training begins. These predefined parameters, known as hyperparameters, play a crucial role in model optimization. Examples of hyperparameters include the learning rate, number of hidden layers, and number of nodes per layer, among many others. Furthermore, the batch size, along with the number of epochs used in the training, also impacts the result.

2.1.2 Fully Connected Neural Networks

Fully connected neural networks (FCN), also known as feedforward neural networks (FNN) or multi-layer perceptrons (MLP), are fundamental architectures in the domain of machine learning. In FNNs, each node in one layer is connected to every neuron in the next layer. Figure 2.1 is an example of an FNN. It consists of one input layer, one or more hidden, and an output layer. The main computation is a linear transformation of the inputs to each layer. FCNs are suitable for a wide range of tasks, including regression and classification.

2.1.3 Loss Function

The loss function plays a crucial role in guiding the neural network's learning process. After the forward pass has been executed, the loss is calculated. The value of the loss represents how well the predictions match the true labels and the choice of loss function varies depending on the specific task within the model's problem domain.

Models aimed at classification tasks predict a probability of each class within the class domain. This makes cross-entropy a well-suited loss function for such tasks, as it uses logarithm to amplify the penalty for confidently incorrect predictions. By doing so, cross-entropy loss encourages the model to produce more accurate and confident probabilities. The cross-entropy loss function is defined as

$$\begin{aligned} \mathcal{L}(\hat{y}, y) &= \frac{\sum_{n=1}^{N} l_n}{N} \\ \text{where } l_n &= -\sum_{c=1}^{C} \log \left(\frac{e^{\hat{y}_{n,c}}}{\sum_{i=1}^{C} e^{\hat{y}_{n,i}}} \right) y_{n,c} \ , \end{aligned} \tag{2.5}$$

where predictions are denoted as $\hat{y}$ and target values as y. N spans the batch dimension, C is the number of classes. Moreover, Focal Loss aims to address class imbalances and is an expansion on cross-entropy loss, see [19] for further reading.

Another popular loss function is the Mean Absolute Error, also called L_1 loss, which calculates the mean absolute error between predictions and target values, making it robust to outliers and less sensitive to large errors. The L_1 loss function is defined as

$$\begin{aligned} \mathcal{L}(\hat{y}, y) &= \frac{\sum_{n=1}^{N} l_n}{N} \\ \text{where } l_n &= |\hat{y}_n - y_n| \end{aligned} \tag{2.6}$$

and N spans the batch dimension.

The intersection over union (IoU) is used to describe the extent of overlap of two boxes [27]. It is defined as

$$\text{IoU} = \frac{|\text{A} \cap \text{B}|}{|\text{A} \cup \text{B}|} \ , \tag{2.7}$$

where A is the ground truth bounding box and B is the predicted bounding box.

2.1.4 Optimizer

Optimizers play an important role in the training process, as they adjust the model's parameters to minimize the loss and enhance the performance. In the training process, gradients are computed with respect to the loss function. These gradients represent the direction and magnitude of changes needed in the model's parameters to minimize the loss. These will guide the optimization process towards searching for the optimal parameters that minimize the loss of the model. An important part of the optimization is the learning rate, which defines the magnitude of the parameter updates in each iteration. The learning rate ranges between zero and one. Higher learning rates can lead to a faster convergence, but they can also cause overshooting and not finding the optimal solution. Too small learning rates, however, can either converge very slowly or encounter the vanishing gradient problem, where gradients become too small and can not effectively update the parameters.

There are different types of optimizers, and the choice can significantly impact the training. Gradient descent is an optimization algorithm based on the principle of minimizing a function by iteratively moving closer towards the minimum value of the function [28]. There are different types of gradient descent optimizers, where the standard gradient descent is defined as

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta) \ . \tag{2.8}$$

θ represents the parameters of the model and $t+1$ demonstrates that the parameters are updated with respect to the current parameters. η is the learning rate, and $\nabla_\theta J(\theta)$ is the gradients of the loss with respect to the model parameters.

Stochastic gradient descent (SGD) is one of the variants of gradient descent. Unlike standard gradient descent, which calculates the gradients with respect to the entire dataset, SGD performs the parameter updates for each data point in the dataset individually. This makes SGD more memory-efficient and faster, especially for large datasets, as computing gradients for the entire dataset is often infeasible. The SGD update rule is defined as

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \ , \tag{2.9}$$

where $x^{(i)}$ is the i-th input to the model, and $y^{(i)}$ is the i-th label or target.

It has been shown that SGD is sensitive to hyperparameters which has made another method, Adam [16], popular. Adam estimates the first and second moments of the

momentum and accounts for those. The result is an optimizer that tends to require less parameter tuning.

2.1.5 Evaluation Metrics

When developing a machine learning model, it is essential to evaluate its performance, especially when comparing it to earlier versions of the model or alternative models. This is done by evaluating the model performance on unseen data, usually called evaluation data, containing target data. Calculating the loss function used in the training on the evaluation data can also be useful, mainly to identify overfitting on the training dataset. However, other useful metrics indicate the network's performance of the network but are not differentiable. These metrics can not be used as loss functions as differentiability is a requirement for backpropagation.

Depending on the task, various evaluation metrics offer insights into its effectiveness by comparing predictions from unseen evaluation data to its target values. Mean average precision (mAP) is a metric used to measure the performance of object detection models by considering both precision and recall. Precision measures how accurate the predictions are, i.e., the percentage of correct predictions among all predictions made by the model. Furthermore, recall measures the proportion of correct predictions among all ground truth instances [11]. To calculate mAP, a criterion on whether a box is a match or not must be set. This is usually done by IoU as defined in 2.7, but there are methods where there is a distance-based criterion instead [3]. The mAP is then calculated by taking the area under the precision-recall graph, which is precision plotted over recall.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs), commonly referred to as ConvNets, are a deep learning neural network architecture used to operate on image data. They have significantly influenced the field of computer vision tasks due to their great ability to capture features and spatial dependencies within images. CNNs are composed of various layer types, with convolutional layers being the core components. These layers operate by sliding a kernel, or filter, across the input image, computing the product between the kernel and localized regions of the image. An overview of a convolution step can be seen in Figure 2.2. The kernel, comprising learnable parameters, facilitates extracting features from the input data. This process yields feature maps, or activation maps, which represent the presence of specific patterns or features within the input data [5] [25].

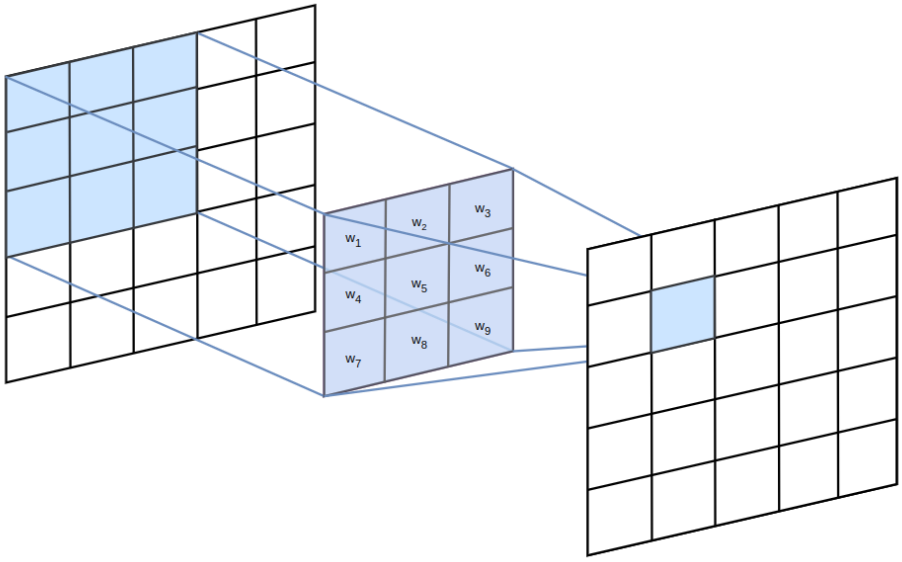


Figure 2.2: An overview of how convolution works is illustrated. The kernel, shown in the middle, is multiplied with a localized region of the input image on the left. This region is known as the receptive field. The kernel, which contains learnable parameters, slides over the image, and the resulting products are summed to produce the output feature map on the right.

Two key characteristic of convolutional layers are their locality awareness and translation-invariant nature. CNNs assume pixels that are close to each other are more related than pixels further away, meaning that local pixel groups often form meaningful patterns which can be detected by the same filter across different image regions. Their translation-invariant nature enables them to detect patterns regardless of their position, rotation, scale, or translation within the image [17].

In contrast to fully connected networks, CNNs share parameters among computations making them more parametrically efficient, which facilitates the creation of larger models. Furthermore, some convolutional layers include downsampling operations, which reduce the spatial resolution while increasing the receptive field. This enables the network to capture more abstract and high-level features in deeper layers. CNNs hierarchically build up features bottom up, where early layers detect simple features like edges and textures, while deeper layers capture more complex patterns and objects. Through the stacking

of convolutional layers, CNNs have the capability to learn increasingly intricate features, making them efficient across a wide range of computer vision tasks [17] [25].

2.2.1 Residual Networks

Residual Networks (ResNets) [12] enhance the basic architecture of CNNs by introducing skip connections, also known as residual connections. These connections enable the training of much deeper networks while improving performance by allowing the activations of a layer to bypass one or more layers and be directly fed to a deeper layer in the network. Through residual learning, layers within ResNets are trained to learn residual functions, focusing on capturing the differences between the desired output and the input data rather than attempting to learn the entire mapping. The residual function can be defined as

$$\mathcal{F}(x) := \mathcal{H}(x) - x \ , \tag{2.10}$$

where $\mathcal{F}(x)$ is the residual function, $\mathcal{H}(x)$ is the desired output, and $x \in \mathbb{R}^{D \times H \times W}$ is the input, with D feature channels and spatial size $H \times W$.

This approach alleviates the degradation problem encountered in very deep networks, where adding more layers leads to decreased performance, by enabling layers to refine the input data rather than reconstructing it entirely. The skip connections form residual blocks, which are the fundamental building blocks of ResNets, as seen in Figure 2.3. By stacking these residual blocks together, ResNets facilitate the flow of information through the network and help mitigate the vanishing gradient problem, which is a common issue in training very deep neural networks. This architecture enables the training of very deep neural networks without sacrificing performance, making ResNets highly effective in various tasks such as image classification and object detection [12].

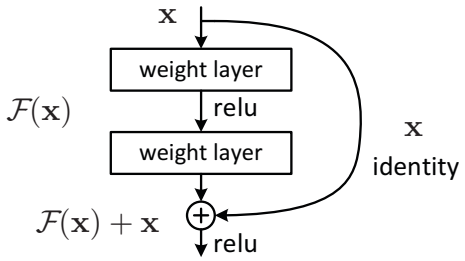


Figure 2.3: Image from [12] shows a residual block where $\mathcal{F}(x)$ is the residual function and x is the input to a layer. The shortcut connection performs identity mapping over two layers.

2.2.2 Deep Layer Aggregation

In CNNs, the distribution of spatial and semantic information varies across different layers. As written in section 2.2, some blocks reduce the spatial resolution by downsampling while increasing the receptive field. This results in earlier layers preserving more spatial details, whereas deeper layers contain less spatial information but richer semantic features. Deep layer aggregation (DLA) serves to integrate both semantic and spatial information across various layers within the network, combining depth, resolution, and scale [32]. This is particularly important in object detection, as it is crucial to localize objects of varying sizes and positions within the image.

Iterative deep layer aggregation (IDA) represents one approach to deep layer aggregation. It follows a sequential stacking, where neighbouring stages are interconnected. A stage consists of a set of blocks operating at a particular resolution. The fusion process starts from high-resolution stages and progressively extends to deeper layers, gradually deepening and refining spatial information within the network. This results in an output that is more semantically rich and higher in resolution. Through the iterative aggregation of stages, IDA effectively learns a comprehensive fusion of both low- and high-level features. This is achieved through a combination of interpolation (upsampling), projection, and convolutional operations. The interpolation and projection parameters are learned during network training [32]. Figure 2.4 depicts the concept of IDA, illustrating the iterative fusion process within a CNN architecture.

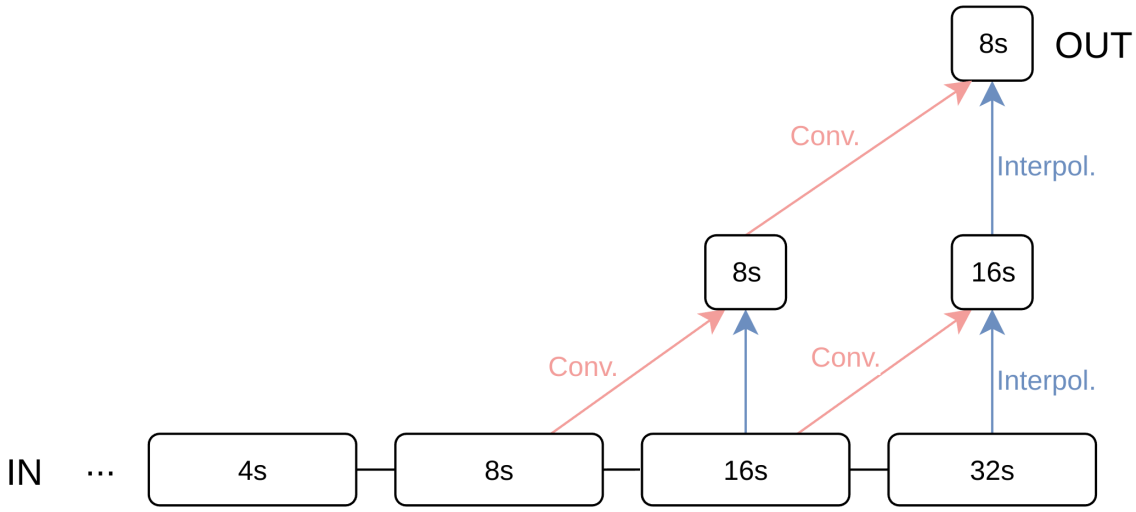


Figure 2.4: The figure illustrates an example of IDA. The wider boxes represent various stages in a ResNet or CNN network, with the input stage at full resolution and subsequent stages progressively downsampled, up to 32 times in the final stage. Convolutional steps are depicted by pink arrows and interpolations by blue arrows. Aggregation nodes, shown as smaller boxes, combine features from different stages. The resulting output is downsampled by a factor of 8 compared to the input image.

2.3 Transformers

A Transformer is a deep learning architecture based on attention mechanisms, allowing the model to draw global dependencies between input and output data [30]. Transformers come in various types tailored to address diverse tasks, ranging from natural language processing (NLP) to image analysis.

The input to a Transformer is divided into tokens and transformed into token embeddings. In NLP, these tokens often represent parts of words, whereas, for image processing, tokens correspond to pixels or patches of pixels. Essentially, what Transformers do is update and refine the content of vectors through a mechanism called attention. There are two types of attention: self-attention and cross-attention. These will be discussed in the following subsections. Given that this project revolves around transformer-based architectures for 3D object detection of images, this theory section will focus on the application of transformers in image processing.

2.3.1 Attention

Attention is the mechanism within the Transformer that maps two inputs to one output. The input data is represented as vectors, which are linearly transformed into matrices called queries, keys, and values. The input is either one or two vectors, depending on whether it is self-attention or cross-attention; this will be discussed in the following sections. The attention used in this project is called scaled dot-product attention, which is defined as

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad . \tag{2.11}$$

Q, K, and V represent the queries, keys, and values, respectively and d_k is the dimension of the query and key. Figure 2.5 shows a visualization of scaled dot-product attention.

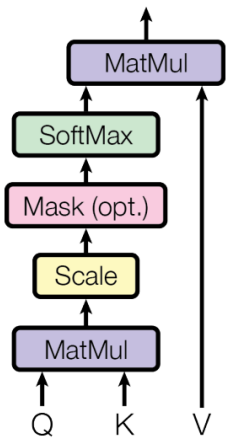


Figure 2.5: Visualization of scaled dot-product attention from the paper [30]. This is achieved by first computing the so-called attention score, which is derived from the dot product of the queries and the transposed keys. The attention scores quantify the strength of the relationship between elements in the input or inputs. Masking can be used to prevent attention between unwanted attention connections. Then, these scores are scaled by dividing them by the square root of the dimension of the key vector and passed through a softmax function. Lastly, the dot product of the value vectors and the output of the softmax function are calculated.

2.3.1.1 Self-Attention

In self-attention, the attention is performed on the same input. This essentially means that each element in the input sequence can attend to other elements within the same sequence, allowing the model to capture relationships and dependencies within the data itself. The attention is computed as in Equation 2.11, where the queries, keys, and values are all linear transformations of the input itself.

2.3.1.2 Cross-Attention

In contrast to self-attention, cross-attention performs the attention on two different data sequences. This enables the model to learn dependencies and relationships between elements of the two inputs. In cross-attention, the attention scores represent the compatibility between the queries from one input and the keys from the other input. It's essential to note that, for the dot product between the queries and the transposed keys, the size of the last dimension in the queries must match the size of the last dimension in the keys. From Equation 2.11, cross-attention can be written as

$$\text{Cross-attention}(Q_1, K_2, V_2) = \text{softmax}(\frac{Q_1 K_2^T}{\sqrt{d_k}})V_2 \ . \tag{2.12}$$

Here, the subscripted digits of Q, K, and V represent the two inputs, clarifying that the Q_1 originates from one input while K_2 and V_2 stem from the other.

2.3.1.3 Multi-Head Attention

Multi-head Attention is an extension of the attention mechanism where the attention is applied multiple times in parallel, each with its own set of learnable parameters. This allows each query to capture different patterns within the input data, providing a richer representation of the information. In contrast to regular attention stated in Equation 2.11, in which queries, keys, and values are processed separately, the multi-head attention performs linear projections on the queries, keys, and values separately for each head before conducting attention in parallel [30]. Multi-head attention is defined as

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \ . \end{aligned} \tag{2.13}$$

The outputs from the regular scaled dot-product attentions are concatenated and fed through another linear projection denoted as W^O. W^Q, W^K, and W^V are the linear parameter matrices for queries, keys, and values, respectively, and the number of heads is denoted as h.

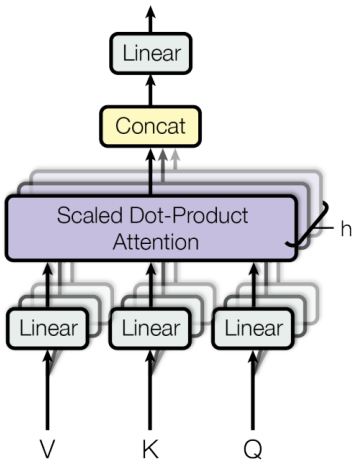


Figure 2.6: Visualization of multi-head attention from the paper [30], where the number of heads is denoted as h.

2.3.2 Positional Encoding

Transformers are inherently permutation invariant, meaning they lack built-in mechanisms to understand the sequential order of input tokens, unlike CNNs that naturally capture the order. To provide the necessary sequential or spatial context to the model, positional encodings are used to describe the information about the positions of the tokens [18].

Coloured images can be represented as a three-dimensional matrix, where two dimensions correspond to height and width, and the third dimension is the channel dimension. The channel dimension describes each pixel in terms of its Red, Green, and Blue (RGB) values. To add positional encoding to an image, positional information is appended to the channel dimension. This means each pixel's position is encoded along with its RGB values, enabling the Transformer to understand the spatial relationships between pixels within the image [18].

2.3.3 Computational Complexity

As machine learning models become deeper and more complex, the time it takes for the model to execute increases. It also requires more resources in terms of memory and computational power. This causes challenges, particularly in the automotive industry, where real-time processing and resource constraints are critical factors. Within the area of computer science, this is referred to as computational complexity and can be categorized into two key areas: time complexity and space complexity.

Time complexity describes how fast the model performs computations while space complexity is related to memory usage and is a measure on the amount of extra memory required to execute the algorithm. The Big O Notation, standing for the order of magnitude, is a measure used to describe computational complexity relative to the input size (n). It provides a way to express how the runtime or space requirements of an algorithm grow as the input size increases, known as asymptotic growth.

Self-attention, represented in Equation 2.11, has complexity $O(n^2 \cdot d)$ per layer, where n is the input size and d is the representation dimensionality of the data. This implies that self-attention scales quadratically with the number of inputs källa(attention is all

you need). In cross-attention, the operation is performed on two different inputs as seen in Equation 2.12. The computational complexity is therefore $O(n \cdot m \cdot d)$ per layer, where m is the size of the other input.

2.4 Perceiver

The paper "Perceiver: General Perception with Iterative Attention" [13], introduces the Perceiver architecture, which is based on Transformers and specifically designed to perform perception tasks. The Perceiver architecture has previously not been evaluated on 3D object detection or for handling large input sizes. Its prior applications have been limited to tasks such as image classification, involving datasets with only 50 000 pixels. Furthermore, the original paper does not include a detailed study of sensor fusion across different modalities. The following section will explain the model architecture and its properties.

2.4.1 Architecture

The Perceiver architecture consists of two main components that the model applies iteratively: a cross-attention module and a Transformer tower. Figure 2.7 illustrates the architecture, showing the iterative application of the cross-attention module and the subsequent Transformer tower. Here, the byte array represents the input data, consisting of pixels in the case of camera images. The size of the byte array is therefore dependent on the input data. The latent array in Figure 2.7 serves as a learnable, condensed array of information derived from the inputs. The fixed-sized latent array that can be chosen to be much smaller than the input byte array, forms an attention bottleneck to reduce computational complexity.

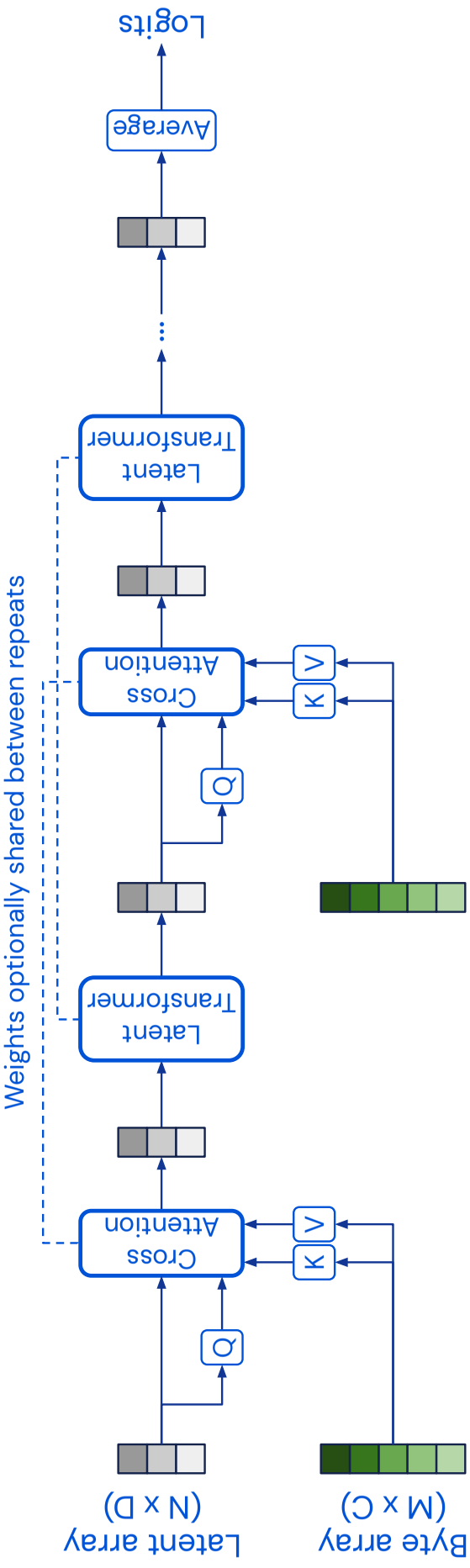


Figure 2.7: An overview of the Perceiver architecture [13], which iteratively applies a cross-attention module followed by a Transformer tower. The byte array is the input data, and the latent array is a smaller learnable array where information from the inputs is stored.

In the cross-attention module, the input byte array and a hidden, learnable latent array are cross-attended to produce an updated latent array. During this step, queries are derived from the latent array while keys and values come from the byte array. This enables the latent array to learn dependencies and relationships from the input byte array. The latent array can be seen as a set of feature vectors that can gather information from the byte array through the cross-attention step.

Following the cross-attention module, the Transformer tower further processes the latent array through several self-attention steps, particularly multi-head attention with eight heads. Here, each latent position exchanges information with the others. By projecting the higher-dimensional byte array to the smaller latent array and then further processing, the latent array captures a comprehensive representation of the input data.

The weights of the cross-attention modules can optionally be shared across all instances, and similarly, the weights can also be optionally shared between each instance of the Transformer tower. Sharing weight across these modules increases the parameter efficiency of the model and can improve generalization by enforcing consistency in how information is processed at different stages [13].

2.4.2 Properties

Among the key properties that distinguish the Perceiver architecture are its scalability and ability to handle arbitrary input data without needing modality-specific architecture changes.

2.4.2.1 Computational Complexity

The Perceiver architecture is designed to handle large-scale datasets efficiently. This efficiency is particularly crucial given the often large size of the byte array, which directly corresponds to the input data. Conversely, the size of the latent array is a hyperparameter and can be chosen to be much smaller than the input data. One of the key factors enabling Perceiver's scalability is its approach to handling the computational complexity of self-attention. Other transformer-based architectures that can solve perception tasks, such as Vision Transformer [9], typically employ self-attention on the (preprocessed) input data, resulting in a quadratic complexity with respect to the input size. However, by mapping the byte array to a latent array of fixed size, Perceiver achieves linear scaling with the input data size. This linear scaling not only enables the construction of deep networks but also ensures efficient training, as opposed to the quadratic complexity associated with standard transformers. Thus, Perceiver facilitates the construction of deep networks while maintaining computational efficiency, making it suitable for various input sizes and tasks [13].

2.4.2.2 Multimodality

Another fundamental property of Perceiver is its multimodal capabilities, allowing it to process arbitrary input data without the need for modality-specific encoding. This property comes from the flexible architectural blocks that make few assumptions about their inputs. Unlike standard tools for similar tasks that necessitate architecture redesigns whenever the input changes, Perceiver's iterative cross-attention mechanism lets us handle different input data effectively [13]. This is in contrast to CNNs, which have inductive

biases that incorporate locality and translational invariance into their architecture, making them primarily applicable to images. One could expect this flexibility to facilitate sensor fusion, making it suitable for integrating data from multiple sources.

2.5 DETR

The DETR (DEtection TRansformer) framework introduces a new technique for 2D object detection where objects are predicted as sets directly with both class and bounding box predictions [4]. Building upon similar principles, DETR3D [31] extends this approach to 3D boxes. Unlike traditional detectors, DETR avoids hand-designed components like anchor generation or non-maximum suppression, replacing them with a set-based global loss and a transformer-based encoder-decoder architecture that performs bipartite matching during training. However, DETR is known to converge slowly, which has led to many studies attempting to improve its training speed, such as [33] and [20].

The DETR architecture consists of three key components: a CNN backbone for image feature extraction, a transformer encoder-decoder for modelling object relations and global context, and a feed-forward network to generate N object predictions. The output from the encode-decoder is a set of object queries, which are summed with a learnable embedding of the same size. The output from the feed-forward network yields a set of box predictions that undergo both bipartite matching and loss computation. The model architecture can be seen in Figure 2.8.

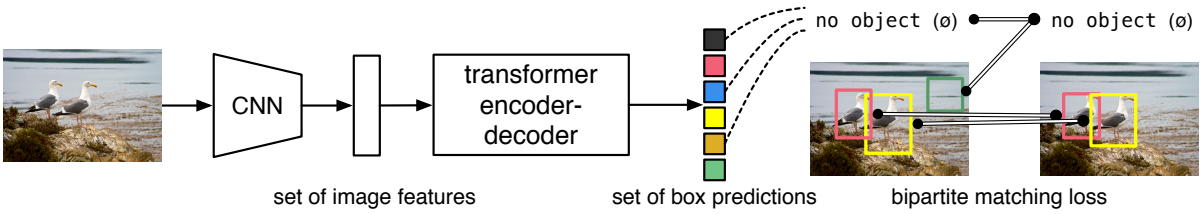


Figure 2.8: Overview of the DETR architecture from the paper [4]. The output from the transformer encode-decoder, the object queries, are fed through a FFN, which generates N predictions. The predictions undergo Hungarian matching followed by loss calculation.

2.5.1 Bipartite Matching and Loss Computation

The number of box predictions the model makes is a fixed number denoted as N, and is set to a number larger than the typical number of objects in the input image. To enable the bipartite matching between ground truth and predicted boxes, the sets need to be of equal size, and therefore, the ground truth set is zero-padded with $\varnothing$ (no object) to size N. The bipartite matching involves finding a representation of unique matches between predictions and ground truths. The matching cost considers both the class labels and boxes. This is done by pairing elements from the prediction and ground truth sets and then finding a permutation with the lowest cost given by

$$\hat{\sigma} = \underset{\sigma \in \mathfrak{S}_N}{\arg\min} \sum_{i}^{N} \mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)}) \ , \tag{2.14}$$

where $\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$ is the pair-wise matching cost between all predictions $\hat{y}$ and all ground truths y. The Hungarian algorithm efficiently finds the optimal one-to-one matching determining the global optimum of the bipartite matching problem in polynomial time.

Once the optimal matching is found, the loss for each prediction to the ground truth

is calculated. This is defined as

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)}) \right] \quad . \tag{2.15}$$

The predicted probability for class c_i is denoted $\hat{p}_{\hat{\sigma}}(c_i)$, where $\hat{\sigma}$ is the optimal assignment calculated in the previous step. The function $\mathbb{1}_{\{c_i \neq \varnothing\}}$ indicates that it becomes zero when the class is $\varnothing$, eliminating the bounding box loss for no objects. Moreover, b_i and $\hat{b}_{\hat{\sigma}(i)}$ is the i-th ground truth bounding box and predicted box respectively. The bounding box loss is defined as

$$\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)}) = \lambda_{\text{iou}} \mathcal{L}_{\text{iou}}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{\text{L}_1} \mathcal{L}_{\text{L}_1}(b_i, \hat{b}_{\sigma(i)}) \quad , \tag{2.16}$$

where λ_{iou} and λ_{L_1} are hyperparameters. The $\mathcal{L}_{\text{L}_1}$ is defined in Equation 2.6, and $\mathcal{L}_{\text{iou}}$ is defined in Equation 2.7.

2.6 Training Optimization

Training deep neural networks requires significant GPU VRAM memory resources and time, presenting a major challenge. Notably, processing images with Transformers demands considerably more compute power than training with CNNs. This project utilizes NVIDIA DGX A100 GPUs [24]. This section presents several strategies aimed at accelerating training and optimizing performance. First, automatic mixed precision training significantly reduces training time and memory requirements. Second, multi-GPU training and data parallelism further accelerate training by distributing computations across multiple GPUs. Additionally, gradient accumulation enables the simulation of larger batch sizes, allowing a more stable training process.

2.6.1 Automatic Mixed Precision

Automatic mixed precision training substantially reduces both memory usage and runtime for deep neural networks. During training, all operations default to single-precision (32-bit). However, many operations can maintain sufficient accuracy with half-precision (16-bit), and are often executed much faster in this format. Automatic mixed precision leverages this by combining both single and half-precision during training [22].

In practice, this means using half-precision for most calculations, while maintaining a single-precision copy of the model parameters to prevent loss of precision. The forward and backward passes are performed in half-precision, significantly reducing memory usage and increasing computation speed. The half-precision gradients are then used to update the optimizer's state, which is kept in single-precision, and these updates are applied to the model parameters stored in single-precision. Finally, this updated single-precision model parameter copy is synced with the half-precision model parameter copy. This process is then repeated for every parameter update [22].

This allows for a significant reduction in training time and memory consumption while preserving the accuracy of the model. In practice, training with automatic mixed precision yields training speeds that are up to three times faster, although the degree of speedup varies depending on factors such as GPU architecture [7].

2.6.2 Multi-GPU Training

Multi-GPU training accelerates the training process by distributing computations across multiple GPUs through parallel processing. Unlike the traditional sequential approach of using a single GPU, parallel processing allows multiple GPUs to simultaneously process different parts of the data [6].

In this project, data parallelism is employed as the parallel processing method. Data parallelism involves duplicating the model across multiple GPUs, with each GPU responsible for processing a subset of the dataset concurrently. Each GPU trains its own copy of the model on its allocated subset of the data. Once the computations are complete, the results from each model are combined, and the training process continues. This approach effectively accelerates training by leveraging the computational power of multiple GPUs [6].

2.6.3 Gradient Accumulation

The batch size influences the training process by affecting the variance in the gradient estimation used to update the model at each iteration [26]. However, as models grow in complexity and size, GPU memory constraints often limit the maximum batch size that can be processed on a single GPU. Memory usage is impacted by batch size because each batch requires storing the input data, activations, gradients, and other intermediate computations in memory. Larger batch sizes increase the memory required to hold all these components simultaneously, leading to higher overall memory consumption [23]. When the batch size is too small, the gradients can exhibit high variance, potentially leading to slower convergence or no convergence at all.

Gradient accumulation functions as an effective strategy to overcome the constraints imposed by limited GPU memory on batch size [23]. Accumulating gradients across multiple smaller batches enables the simulation of larger effective batch sizes. Throughout the training process, the dataset is divided into smaller batches. For each batch, the loss is computed, followed by the calculation of gradients. These gradients are then accumulated over a specified number of successive batches without immediately updating the model parameters. During gradient accumulation, the gradients are stored rather than storing all intermediate activations to avoid running out of memory. After the specified number of batches, the accumulated gradients are used to update the model parameters, ensuring a more stable optimization process. The updated model parameters using gradient accumulation and stochastic gradient descent are calculated as

$$\theta_{t+1} = \theta_t - \eta \cdot \sum_{j=1}^{M} \nabla_\theta J(\theta; x^{(ij)}; y^{(ij)}) \tag{2.17}$$

where θ_t represents the current model parameters, η is the learning rate, M is the number of batch accumulations and $\nabla_\theta J(\theta; x^{(ij)}; y^{(ij)})$ is the gradients of the loss with respect to the model parameters for the i-th data point in the j-th mini-batch. This results in a larger effective batch size, which is equal to the batch size multiplied by the number of batch accumulations [23].

2.7 Camera and LiDAR Projection

Understanding the relationship between the physical world and the data captured by sensors is crucial in computer vision tasks. The data in this project are obtained from both LiDAR and camera, as written in Section 3.1. This section explores the process of projecting data between different reference frames and spaces, specifically focusing on projection from 3D to 2D.

2.7.1 Extrinsics and Intrinsics

The extrinsic and intrinsic matrices are used to project data between reference frames and spaces. The extrinsic parameters describe the frame's position and orientation in the reference coordinate system. The extrinsic camera matrix describes the transformation between the reference frame and the camera frame, containing the camera's rotation and translation relative to the reference frame. Similarly, the extrinsic LiDAR matrix describes the LiDAR's rotation and translation relative to the reference frame. These transformations are crucial for integrating data from the two sensors and aligning them within a common frame of reference [29].

To convert a 3D point in the LiDAR frame to a 3D point in the reference frame, homogeneous coordinates are used. This transformation can be calculated as follows

$$\mathbf{p}_r = \mathbf{T}_l \mathbf{p}_l \ , \tag{2.18}$$

where $\mathbf{p}_r$ is the 3D point in the reference frame in homogeneous coordinates, $\mathbf{p}_l$ is the 3D point in the LiDAR frame in homogeneous coordinates, and $\mathbf{T}_l$ is the LiDAR extrinsic matrix. The LiDAR extrinsic matrix is defined as

$$\mathbf{T}_l = \begin{bmatrix} \mathbf{R}_l & \mathbf{t}_l \\ \mathbf{0}^T & 1 \end{bmatrix} \ , \tag{2.19}$$

where $\mathbf{R}_l$ is the rotation matrix, $\mathbf{t}_l$ is the translation vector, and $\mathbf{0}^T$ is a 1x3 zero vector. Similarly, the 3D point in the reference frame can be projected to the camera frame by multiplying the homogeneous point with the inverse of the camera extrinsic matrix, as follows

$$\mathbf{p}_c = \mathbf{T}_c^{-1} \mathbf{p}_r \ . \tag{2.20}$$

where $\mathbf{p}_c$ is the 3D point in the camera frame in homogeneous coordinates and $\mathbf{T}_c$ is the camera extrinsic matrix.

The intrinsic camera matrix, also known as the calibration matrix, contains the internal parameters of the camera. The camera matrix enables mapping a 3D point in the camera frame to a 2D point in the image plane. The intrinsic camera matrix, $\mathbf{K}$, can be defined as

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \ , \tag{2.21}$$

where f_x and f_y are the focal lengths in the x and y dimensions, s represents any skew between the sensor axes, and c_x and c_y are the coordinates of the image center. Using the pinhole camera model, a matrix multiplication between the intrinsic matrix and a homogeneous 3D point in the camera frame yields a 2D homogeneous point on the image plane. This is calculated as

$$\tilde{\mathbf{x}}_s = \mathbf{K} \mathbf{p}_c \ , \tag{2.22}$$

where $\tilde{\mathbf{x}}_s$ is a 2D homogeneous coordinate and $\mathbf{p_c}$ is a 3D point in the camera frame in homogeneous coordinates. The 2D homogeneous coordinate $\tilde{\mathbf{x}}_s = \begin{bmatrix} u & v & w \end{bmatrix}^T$ can then be normalized to pixel coordinates $\begin{bmatrix} u' & v' \end{bmatrix}$ as

$$u' = \frac{u}{w}, \quad v' = \frac{v}{w} \ . \tag{2.23}$$

While the pinhole camera model simplifies the projection process, it does not account for lens distortions, especially with wide-angle lenses used in this project. To achieve accurate projection, the Kannala-Brandt model is used to account for these distortions. This model, combined with the camera's intrinsic parameters and distortion coefficients, provides a more accurate representation of the wide-angle lens. See [15] for more information on the Kannala-Brandt model.

3
Data

High-quality data is essential for effective neural network training. This section provides an overview of the dataset utilized in this project. Furthermore, the preprocessing steps implemented to align the dataset with the project's objectives are presented.

3.1 ZOD Dataset

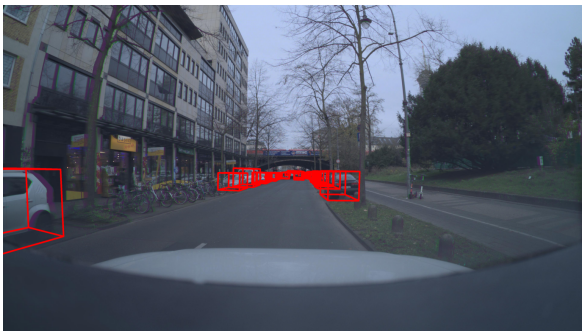
Zenseact Open Dataset (ZOD) is a large-scale and diverse multimodal dataset specifically designed for autonomous driving research [2]. It stands out due to its high-resolution data and extensive annotations, surpassing other comparable datasets. ZOD includes approximately 100,000 annotated camera images, also including additional sensor data and longer sequence recordings.

The data for ZOD is collected across various European countries, which inherently reflects those scenarios' demographic composition and traffic scenarios. Despite the geographic focus, it covers a wide range of weather conditions, lighting variations, and diverse driving scenarios.

For this project, the focus is on camera images with annotated 2D and 3D bounding boxes of vehicles. This makes the dataset useful since it includes detailed 2D and 3D annotations of objects, with object detection ranges extending up to 245 meters. ZOD includes a wide variety of objects, including vehicles. The Vehicle class in ZOD is further subdivided into nine subclasses: Car, Van, Truck, Trailer, Bus, HeavyEquip, TramTrain, Other and Inconclusive. In total, the dataset contains over one million annotated vehicles.

The 2D annotations are represented with a class label and a bounding box with four pixel values: x_{min}, y_{min}, x_{max}, and y_{max}. The 3D annotations also have a class label but contain more data. They have a center coordinate, an orientation in quaternions, both represented in a LiDAR frame, and a size given by height, width, and length. Figures 3.1a and 3.1b show images with 2D and 3D bounding boxes for vehicle objects, respectively.

(a) An image with 2D bounding boxes for vehicle objects.



(b) An image with 3D bounding boxes for vehicle objects.

Figure 3.1: Images illustrating 2D and 3D bounding boxes for vehicle objects.

3.2 Data Preprocessing

Data preprocessing is crucial in preparing the dataset for neural network training. For this project, it involves ensuring that only the necessary data is used to optimize memory usage efficiently and formatting the data correctly.

The raw camera data contains 4K resolution wide-angle images and, therefore, takes up a lot of memory. To reduce memory usage and allow for a larger batch size, which in turn enhances training, the images are both cropped and down-scaled. Removing unnecessary areas, such as sections encompassing the vehicle's hood and large portions of the sky, enables the network to focus on the most relevant features within the images. The region of interest (ROI) is specifically chosen to include the most relevant parts of the images for the task at hand. The chosen ROI was to remove four pixels on each side, 428 pixels on the top, and 588 pixels on the bottom. This essentially removes the hood of the vehicle as well as the horizon. Furthermore, the images are downscaled by a factor of nine resulting in lower-resolution images that maintain sufficient quality for object detection. An example of an image with the original size is seen in Figure 3.2, and a final cropped and downscaled image is seen in Figure 3.3. The original images consist of 8 342 464 pixels and the downscaling and cropping results in a total of 463 872 pixels for each image.

Figure 3.2: An image from ZOD dataset taken with the front camera. This image contains 8 342 464 pixels.

Figure 3.3: Same image as in Figure 3.2, but with the desired ROI and scaled down by a factor of nine. This image contains 463 872 pixels.

The 3D bounding box annotations in the ZOD dataset are represented in the LiDAR frame. Since this project aims to predict 3D bounding boxes from only the image as input, it makes sense to transform the 3D bounding box annotations to the camera frame. Furthermore, objects in the dataset labelled with ”high occlusion level” are removed for this project, as predicting these objects can be particularly challenging.

4

Per3D-ObDet

In the process of adapting Perceiver for 3D object detection, various modifications and additions to the model were implemented. This section will go through both the original components of the architecture and the additional components introduced during the implementation phase of the Perceiver 3D Object Detector (Per3D-ObDet).

4.1 Base model

To make Perceiver predict 3D objects in the input image, an object detection head is added to the original Perceiver architecture. Figure 4.1 illustrates the essential components of Per3D-ObDet. This section provides an in-depth explanation of these components.

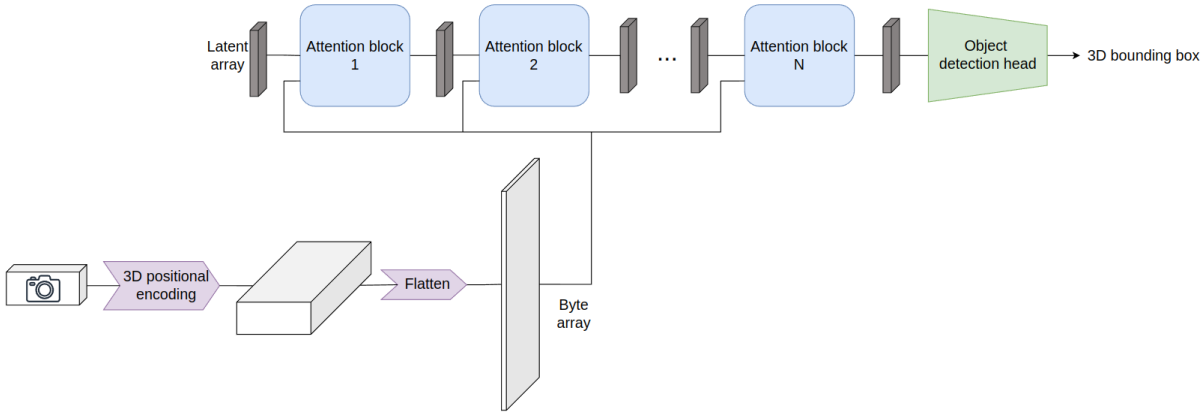


Figure 4.1: Illustration of the essential components of Per3D-ObDet. The latent array, which is learnable, and the byte array are passed through multiple attention blocks where the number of attention blocks is denoted as N. The output from every attention block is an updated version of the latent array. After the last attention block, the latent array is fed through an object detection head that predicts bounding boxes of vehicles in the input image.

4.1.1 Attention Block

The original Perceiver model consists of a cross-attention followed by a transformer tower as seen in Figure 2.7. In this project, this is referred to as an attention block and is depicted in Figure 4.2 for more clarity. This block will be applied iteratively multiple times, with the number of iterations being a hyperparameter.

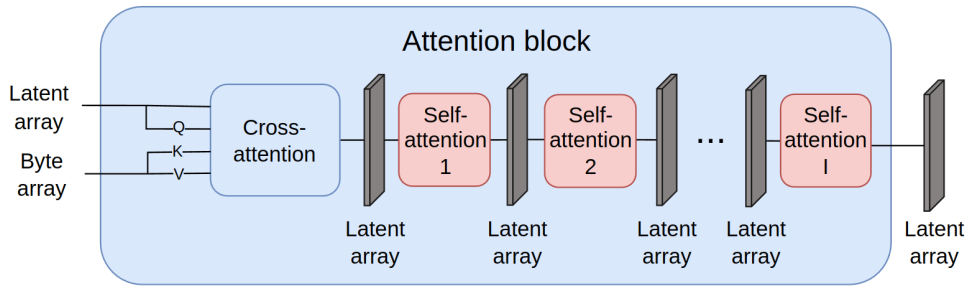


Figure 4.2: Clarification of an attention block, which includes one cross-attention followed by multiple multi-head self-attentions. The number of self-attentions is denoted as I and is a hyperparameter. The inputs to the attention block are the latent array and the byte array, and the output is an updated latent array.

The self-attentions in the attention block have their own weights but can be shared between nearby attention blocks. This configuration is defined by weight-sharing groups, where the groups are sequentially arranged and not spread out. This setup of how the groups are arranged is also another hyperparameter and is defined as a list of integers starting from zero and increasing sequentially, which can include repeated values. The number of attention blocks is the length of this list. Here are some examples of five attention blocks in a sequence:

Table 4.1: Examples of different weight sharing configurations and corresponding explanation. These examples are for a model with five attention blocks in total.

Weight sharing list	Explanation
`[0, 1, 2, 3, 4]`	Every block has its own weights.
`[0, 0, 1, 2, 3]`	The first two blocks share weights, and the rest have their own weights.
`[0, 0, 1, 1, 1]`	The first two blocks share weights, and the three last blocks share weights.
`[0, 0, 0, 0, 0]`	The weights are shared among all blocks.

4.1.2 Object Detection Head

In the original Perceiver model, the latent array from the last attention block is averaged across latent positions into a single vector representing class probabilities. However, the Per3D-ObDet seeks to predict 3D bounding boxes for multiple vehicles in the input image. Therefore, the head of the Per3D-ObDet is inspired by the DETR architecture. The idea is that the latent array represents the object queries of the image, where then every query is fed through small MLPs that generate class and bounding box predictions. They are then matched to the ground truths as in DETR, which will be described later in Section 4.2.3.

This project aims to predict 3D bounding boxes where the 3D annotations consist of a

class label, a center coordinate, a quaternion orientation, and a size. Once the model has generated object queries after the last attention block, they are fed through a class MLP and a bounding box MLP. The number as well as the dimension of the queries, denoted as N and D respectively in Figure 4.3, are hyperparameters. The class MLP consists of linear layers and a Sigmoid activation function given by Equation 2.3. Furthermore, the bounding box MLP also consists of linear layers with ReLU activations on the input and hidden layers. The number of layers is evaluated during the optimization of the model. This small network produces ten outputs: three for the centre, three for size and four for quaternion orientation. The size undergoes a ReLU function that ensures no negative size predictions.

Given that images inherently exist in two dimensions, detecting 2D bounding boxes can be considered a less complex task compared to detecting 3D bounding boxes. Therefore, this project will examine the performance of detecting 2D boxes along with the 3D boxes. The idea is that incorporating 2D predictions may accelerate and enhance the 3D predictions, as finding the correct matches is important. This is done with an additional 2D bounding box MLP consisting of linear layers that output x_{min}, y_{min}, x_{max}, and y_{max}. The dataset used in the project also provides 2D bounding box annotations, enabling the generation of 2D predictions.

4.2 Additional components

This section presents the additional components and modifications introduced to enhance the performance and functionality of the base model architecture in Figure 4.1. These components aim to address specific challenges and optimize the model's ability to accurately perform 3D object detection.

4.2.1 Grouped Cross-Attention

One of the main components of the Perceiver model is its cross-attention mechanism between the byte array and a learnable latent array. The byte array represents the input image, which can become very large for high-resolution images. High resolution is essential for object localization, although it is less critical for image classification. In the original Perceiver paper, where one of the tasks was image classification, the images contained approximately 50 000 pixels. In contrast, the images in this project have approximately 464 000 pixels, even after preprocessing as described in Section 3.2. This significantly increases the computational time on Per3D-ObDet, making it much slower and turning cross-attention into an even greater bottleneck than in the original Perceiver. As the computational complexity of cross-attention is dependent on both input sizes, it would be beneficial to somehow make those inputs smaller. This was done by dividing the inputs into groups and putting that information in the batch dimension of the tensor. This changes the complexity of cross-attention from $O(n \cdot m \cdot d)$ to

$$O(g(m \cdot n/g^2) \cdot d) = O((m \cdot n \cdot /g) \cdot d) \ , \tag{4.1}$$

where g is the number of groups. Figure 4.3 illustrates the grouped cross-attention.

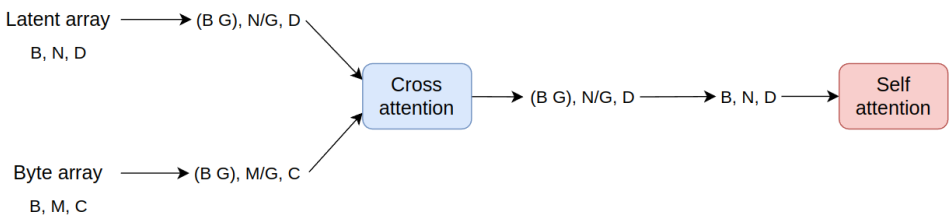


Figure 4.3: Illustration of grouped cross-attention showing how the sizes of the input tensors are modified to make cross-attention more computationally effective. B represents the batch dimension, while N and D denote the number and dimension of the latent arrays. Similarly, M and C represent the size and dimension of the byte array.

When grouping the inputs, different parts of the latent array attend to different regions of the byte array. To ensure each part of the latent array attends to a local pixel group, as vehicles typically appear in localized areas of an image, the byte array (input image) is divided into rectangular sections. This division ensures that specific parts of the latent array focus on the corresponding pixels within these sections. Following this localized cross-attention, information is shared between these pixel groups through self-attention, facilitating the integration of context across the entire image.

4.2.2 Learnable Reference Position Arrays

In the implementation of DETR, a learnable positional embedding is added to the object queries. This concept can also be applied to Per3D-ObDet. A separate learnable array of the same size as the latent array is defined and added to the latent array prior to every attention step. Figure 4.4 clarifies the implementation of this learnable positional embedding.

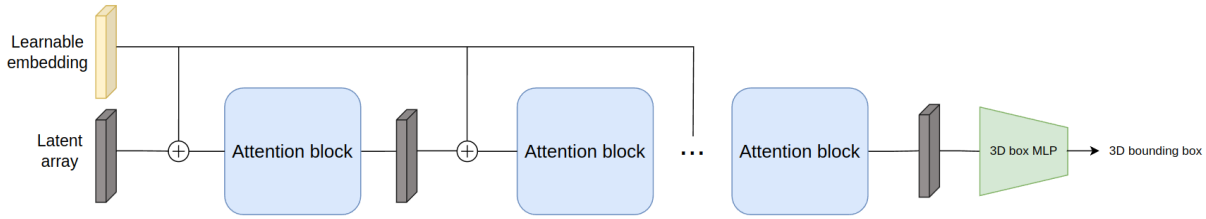


Figure 4.4: Figure showing how the learnable positional embedding is added to the latent array before every attention block. The byte array is excluded to emphasize the latent array and the learnable embedding.

Another idea is to let the model predict an offset to a center reference point rather than predicting the center position directly. The reference position is defined as a learnable array of the same size as the latent array. This would essentially mean that this array learns a reasonable starting point and instead predicts an offset to this reference. Before each attention block, the reference position array is added to the latent array. This array is then fed through an additional MLP where a reference center position is predicted. The reference position MLP consists of linear layers with ReLU activation function and outputs three values representing the reference center coordinates. This predicted reference center is subsequently added to the predicted center from the 3D bounding box MLP.

When the model also predicts 2D bounding boxes, the same principle of a learnable reference position can be applied. In this case, a separate array with the same size as the latent array is defined, along with an additional MLP for predicting the 2D reference box. In the 2D case, the bounding boxes only consist of four values representing the 2D box corners. Hence, the reference position should represent a reference box rather than a center point. Figure 4.5 illustrates the concept of both the 3D and 2D reference arrays.

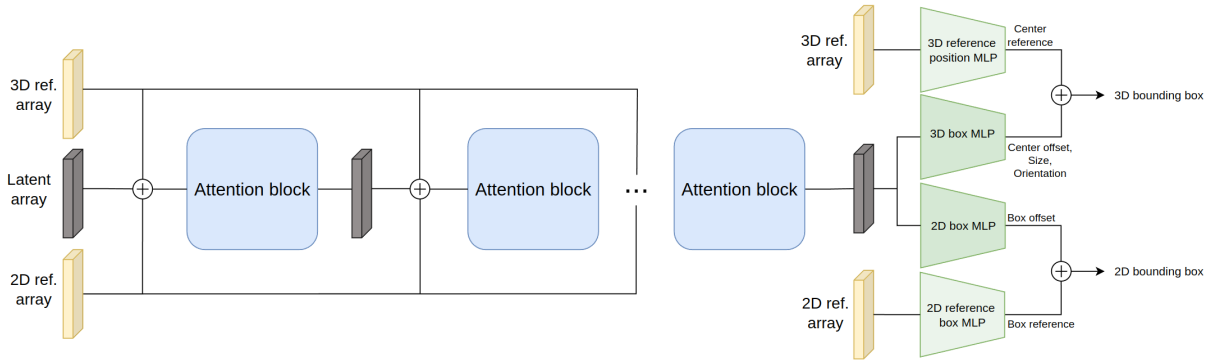


Figure 4.5: Figure showing the concept of the learnable 3d reference position and 2D reference box. The byte array is excluded to emphasize the latent array and the reference arrays. The reference arrays are added to the latent array prior to every attention block. The original arrays are fed through separate MLPs, generating a center reference and a reference box, respectively. The center and box references are then added to the offset predictions to give the final prediction of the bounding boxes.

Another approach, when utilizing 2D predictions on top of 3D predictions, is to use a single reference position array rather than two separate arrays to simultaneously predict the 2D and 3D reference positions. In this case, the reference position MLP outputs seven values: three for the 3D reference center, and four for the 2D reference box. The 3D reference center coordinates are simply added to the 3D offset predictions, and the 2D reference box is added to the 2D offset prediction.

To summarize this section, there are five different approaches to a learnable reference position array:

- Learnable embedding
- 3D reference array
- 2D reference array
- One reference array for 2D and 3D

4.2.3 Matching and Loss Computation

The matching and loss computation of the predictions made by the object detection head is heavily inspired by the matching and loss computation from DETR. For this project, the matching between prediction and ground truth objects is done for class and center.

This project is limited to predicting only vehicles in the image, resulting in a classification space with two classes: vehicle and $\varnothing$ (no object). The maximum number of ground truth boxes, which is also a hyperparameter, is fixed to enable matching between ground truth boxes and predicted boxes. If an image contains more ground truth objects than this fixed number, the objects are sorted by distance, and the furthest ones are removed

to meet the required number of boxes. Conversely, if there are fewer objects than the fixed number, the ground truth boxes are zero-padded with $\varnothing$ (no object).

4.2.3.1 Distance Weights

Detecting vehicles in the image becomes more challenging as it gets further away due to the decreased resolution. At the same time, it becomes more critical to ensure accurate predictions for nearby vehicles, especially in the context of AD and ADAS. To prioritize the detection of nearby vehicles, one approach is to introduce distance-based weighting for the losses. The idea is that after the predictions have been matched to the ground truths, they are given a weight based on the ground-truth distance. For this project, two different functions are used given by

$$\mathrm{w}_1(d) = \min(2, \frac{1}{(d/60)^2}) \tag{4.2}$$

and

$$\mathrm{w}_2(d) = \min(2, \frac{1}{d/50}) \ . \tag{4.3}$$

d is the distance calculated with the Pythagorean theorem to the center point. These decreasing curves will put low weight on objects far away, implying that the loss will be decreased for those objects and thereby make the model put less focus on those objects.

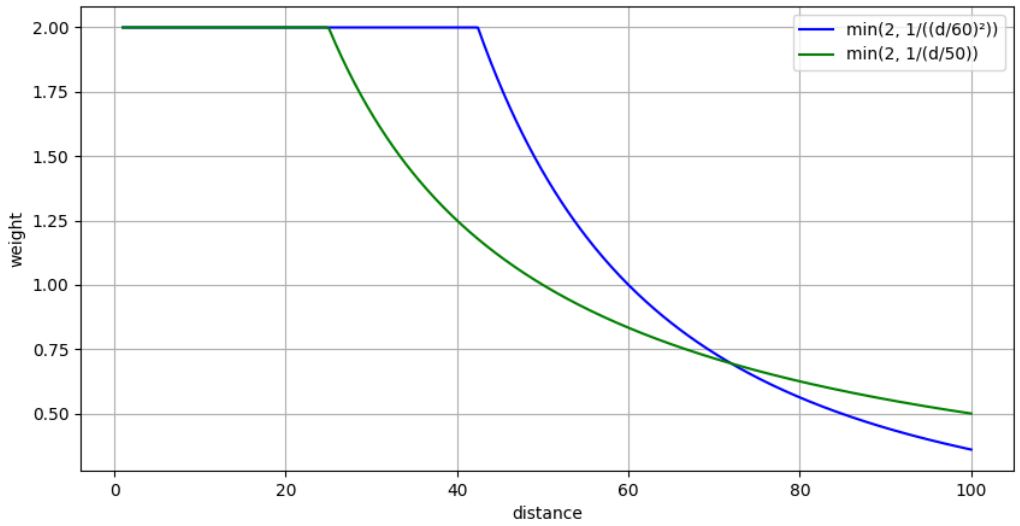


Figure 4.6: Plot of the two distance-based weights.

4.2.3.2 Auxiliary Losses

Auxiliary losses are employed in the process of training Per3D-ObDet to force the model to produce better predictions in the earlier attention blocks and avoid poor initial performance. After each attention block, the object detection head is applied to generate intermediate predictions. These intermediate predictions are used solely for loss calculation, with all auxiliary losses summed together and applied during backpropagation. Figure 4.7 shows how and where the auxiliary losses are extracted from the model.

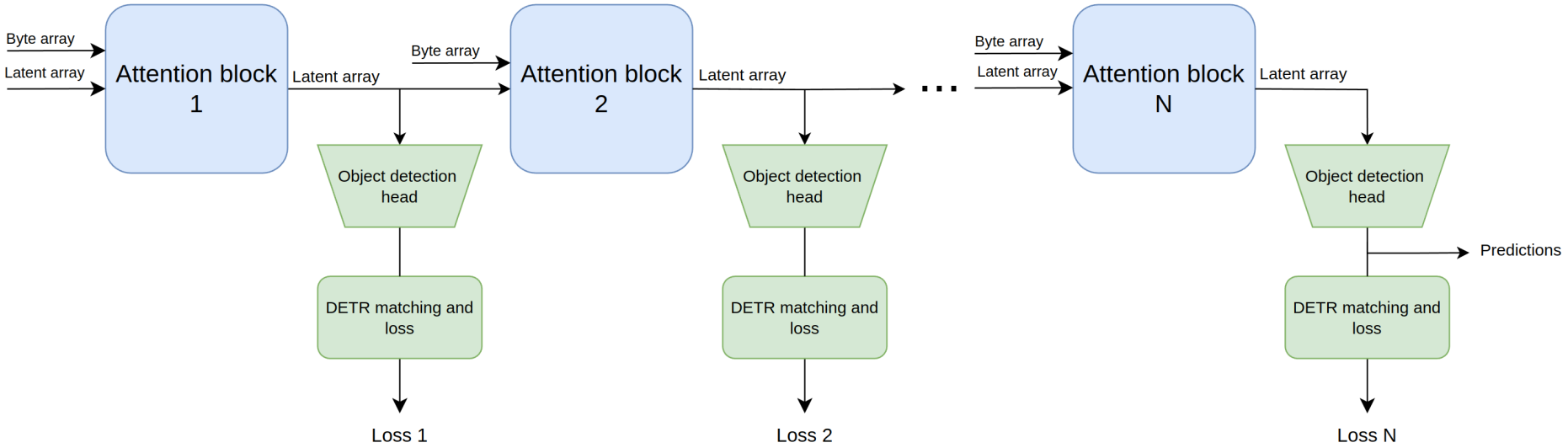


Figure 4.7: Figure showing how the auxiliary losses are extracted where N denotes the number of attention blocks and thus the number of losses computed throughout the network. The latent array, produced by each attention block, is fed into both the next attention block and the object detection head, which generates predictions. These predictions are compared to the ground truths to calculate the losses. After the last attention block, the latent array is fed through the object detection head, generating the final predictions and loss. Before backpropagation, all losses (1, 2, ..., N) are summed together.

4.2.4 Positional Encodings

This project utilizes 3D positional encodings to integrate spatial information into the model. Each pixel in the byte array is assigned 3D coordinates just before the byte array is flattened and fed into the attention blocks. These 3D positional encodings correspond to points along a ray cast from the camera into the scene, representing the pixel's position in the 3D world frame. To derive the 3D world coordinates from the 2D pixel coordinates, the pixel projections are inverted, and multiple points are generated along the depth axis. Specifically, for each pixel in the input image, 3D points are generated representing depths up to 100 meters. This depth range is chosen because the model is only trained to detect vehicles within a 100-meter distance. By encoding these 3D positional coordinates, the model gains a better understanding of the spatial relationships and depth information in the scene.

4.2.5 Backbone

When evaluating the convergence ability of the model, it is desirable to minimize the time per epoch, especially given the project's time constraints. The first step in the Perceiver model is cross-attention, whose computational complexity depends on the input sizes, expressed as $O(n \cdot m \cdot d)$. Hence, one way to make training time faster is to make the height and width of the input image, fed into the cross-attention, smaller. This can be done using a ResNet that progressively downsamples the image. However, to avoid excessive decrease in the height and width of the image, while still keeping semantic information from highly downsampled layers, DLA is used. This results in an image that is downsampled eight times.

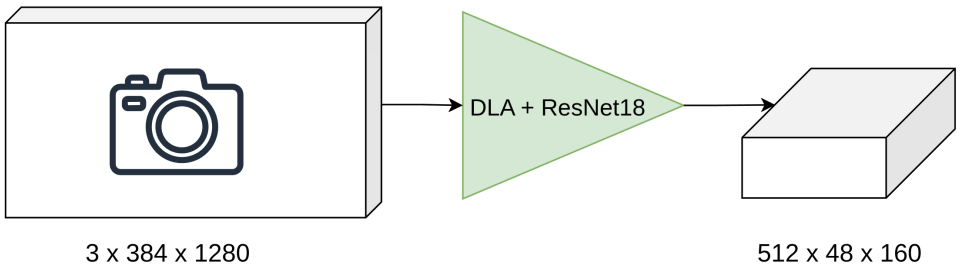


Figure 4.8: Image showing the size of the image that is fed through the DLA with a ResNet18 and the size of its output. The sizes are defined as channels x height x width.

5
Method

Perceiver has never been tested on object detection, making the number of hyperparameters and configurations to check very large. To factorize the search space, it was decided first to identify the critical components for achieving good object detection performance on ZOD with a DETR-based loss function through an ablation study. This was carried out using a ResNet18 backbone to encode images to speed up the experiments and because of its already proven robustness in image processing and object detection in particular. The ResNet18 backbone also downscales the images by a factor of eight, reducing the computational demands of the cross-attention mechanism and speeding up training times. After identifying the best configurations from the ablation study, a hyperparameter study was conducted.

5.1 Ablation Study

The ablation study consists of 13 configurations of Per3D-ObDet, where the idea is to identify the contributions of each additional component to the base model seen in Figure 5.1. The base model is trained and compared to models where each configuration is added individually and in various combinations. While not every combination of the additional components is tested, the selected configurations in this section are expected to highlight the impact of each part. This analysis will guide the project in determining the optimal version of Per3D-ObDet.

All 13 trainings are launched on one GPU each and run for 25 epochs. The base model for the ablation study has a ResNet18 backbone with DLA to make the training process faster. Table 5.1 shows an overview of the additional models, and the caption describes the models more in-depth. The models are categorized into a base model, and the rest are further divided into four groups for easier comparison, where each group has its own focus area from Section 4.2.

To identify the optimal configurations, the results are analyzed through plots of mAP losses with respect to epoch iterations. The mAP is calculated for a separate evaluation dataset from ZOD and indicates how well the models are performing as training time increases.

Additionally, a training session without the ResNet backbone is conducted using the final configurations from the ablation study to evaluate the model's performance under real-world conditions. The hyperparameters are set to reasonable values. Only one training session without the backbone is performed, as each epoch takes significantly longer without the backbone to speed up the training process, consuming considerable computational and time resources. However, testing without the backbone provides valuable insights because it preserves one of the main properties of Perceiver, namely its multimodality.

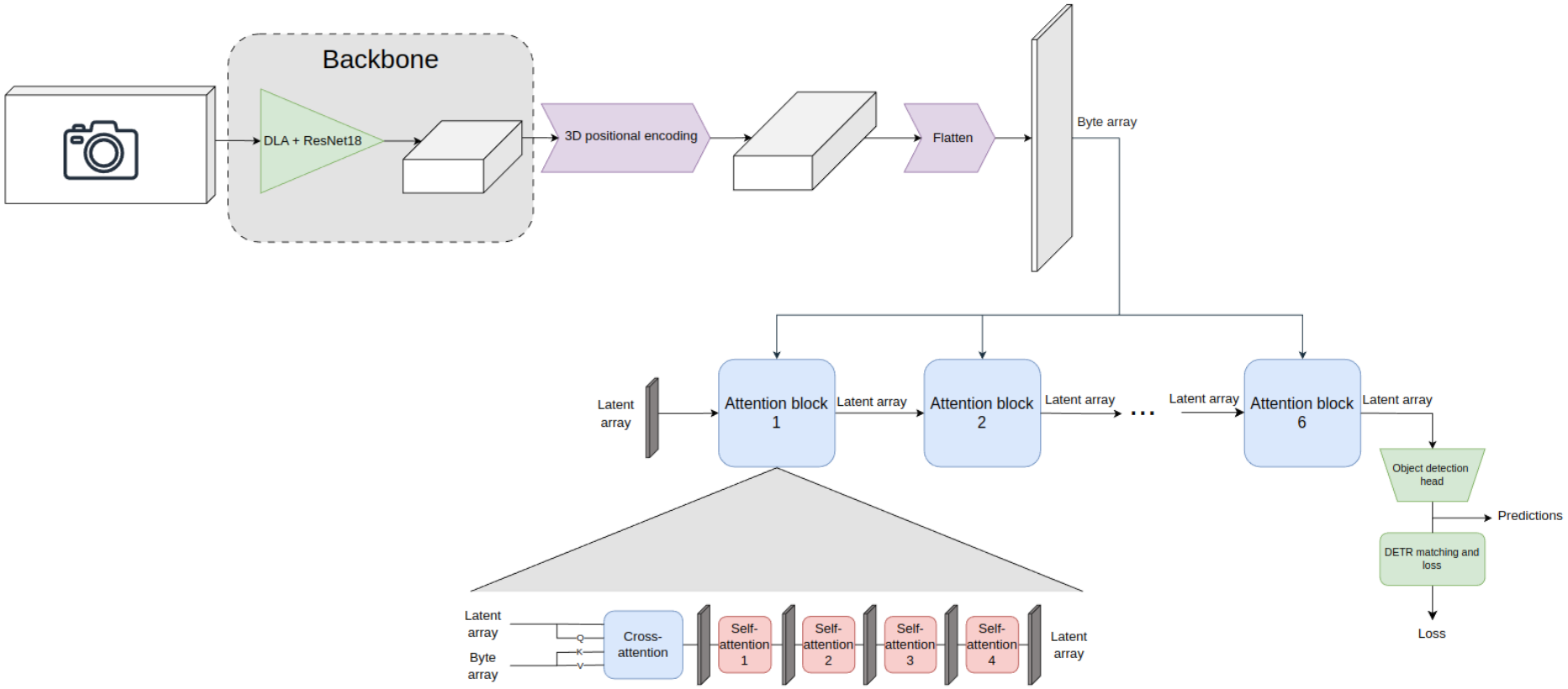


Figure 5.1: Illustration of the base model for the ablation study, consisting of only the necessary parts of Per3D-ObDet. The input image passes through the backbone, which is a pre-trained ResNet18 with DLA. The output is then assigned 3D positional encodings, after which it is flattened to form the byte array. The same byte array is fed into each attention block along with the learnable latent array, which is updated after each attention block. The attention blocks have their own weights, and the first attention block is shown in detail. The blocks consist of one cross-attention followed by four self-attentions that all have their own weights. After six iterations of attention blocks, the latent array is fed through the class MLP and the 3D bounding box MLP. These predictions are then passed to the matching and loss computation.

Table 5.1: Overview of the ablation study where the checkmarks mark what configurations are used in the different models. All models use a pre-trained ResNet18 backbone with DLA and the 3D positional encodings. The 3D and 2D reference arrays, along with the one reference array and the learnable embedding are described in Section 4.2.2. The distance weights and auxiliary losses are detailed in Section 4.2.3. The horizontal lines define four groups that focus on similar aspects but with slight differences, allowing for comparison within each group.

Name	**Auxiliary losses**	**3D reference array**	**2D predictions**	**2D reference array**	**One reference array for 2D and 3D**	**Learnable embedding**	**Distance weight**
Base							uniform
A							$w_1(d)$
B							$w_2(d)$
C	✓						uniform
D		✓					uniform
E	✓	✓					uniform
F			✓				uniform
G		✓	✓	✓			uniform
H	✓	✓	✓	✓			uniform
I	✓		✓		✓		uniform
J			✓		✓		uniform
K	✓		✓		✓		uniform
L						✓	uniform

5.2 Hyperparameter Study

Once the optimal model of Per3D-ObDet is identified from the ablation study, a hyperparameter study is conducted. This study also incorporates the ResNet backbone from the ablation study to accelerate the training process.

The hyperparameter study involves running 50 training sessions of Per3D-ObDet with randomly selected hyperparameters within a reasonable range and can be seen in Table 5.2. These models will run for 30 epochs before being evaluated. To evaluate the performance of the trained models, plots are created for each hyperparameter against mAP. These plots illustrate the relationship between values on each hyperparameter and the resulting mAP, highlighting which hyperparameter values yield the best performance.

Table 5.2: Table showing all hyperparameters that are a part of the hyperparameter study. The parameters are either chosen from a uniform distribution or from a list of numbers.

Hyperparameter	**Distribution/Value**
`Learning rate`	[1e-4, 2e-4, 3e-4, 4e-4, 5e-4, 7e-4, 1e-3]
`Number of cross-attention groups`	[4, 16, 64]
`Number of latents`	Integer between 150 and 600 and must be divisible with the number of cross-attention groups
`Latent dimension`	[128, 256, 512]
`Number of attention blocks`	Integer between 1 and 8
`Weight sharing among attention blocks`	List of length `Number of attention blocks` with numbers describing which blocks are sharing weights, as described in Table 4.1. There are maximum three different weight sharing groups
`Number of self-attention steps per attention block`	Integer between 1 and 8
`Number of self-attention heads`	[4, 8]
`Number of cross-attention-heads`	[1, 2, 4]
`Expansion factor self-attention`	[1, 2, 4]
`Dropout rate`	Number between 0 and 0.25
`Matching weight 2D box`	Number between 0.5 and 50
`Matching weight 3D center`	Number between 0.5 and 50
`Class weight`	Number between 5 and 30
`No object weight`	Number between 0.05 and 0.95
`3D center weight`	Number between 0.004 and 0.4
`3D size weight`	Number between 0.03 and 3
`3D orientation weight`	Number between 0.04 and 4
`2D box weight`	Number between 0.6 and 20

6
Results

This section presents the results of the ablation and hyperparameter study presented in Chapter 5, as well as the final model of Per3D-ObDet.

6.1 Results from the Ablation Study

All the models from the ablation study are trained for 25 epochs. The resulting plots of their respective mAP score can be seen in Figures 6.1a, 6.1b, 6.2a, and 6.2b, with the base model plotted in each as a reference. The different model configurations are detailed in Table 5.1.

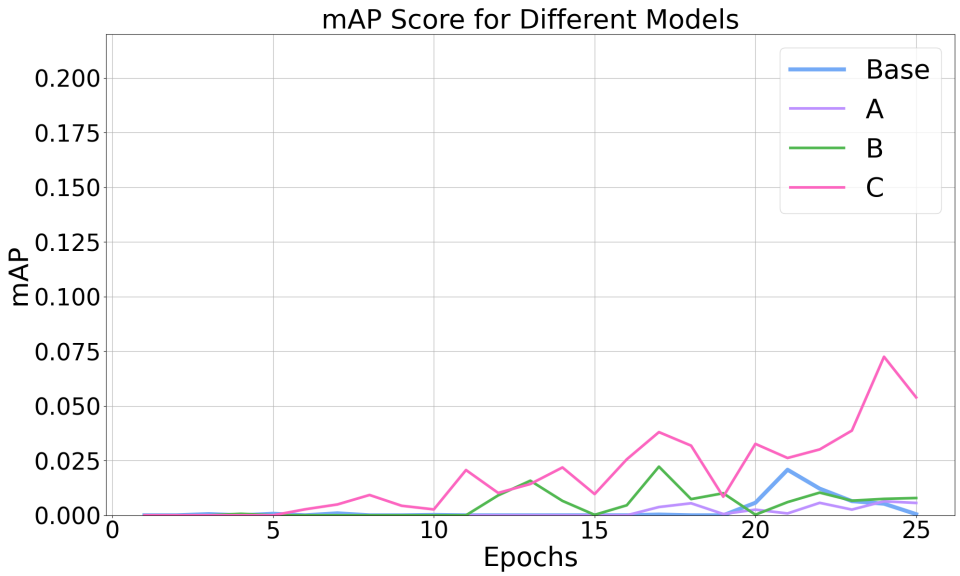
From Figure 6.1a, it is evident that adding auxiliary losses to the base model improves performance, as model C achieves the highest mAP scores. Models B and C incorporate different weights on the objects depending on distance, assigning higher weights to nearby vehicles using Equations 4.2 and 4.3, respectively. Comparing models A and B to the base model shows no significant differences, although model B performs slightly better than both the base model and model C.

The next group compares the impact of incorporating a 3D reference array with versus without auxiliary losses. As shown in Figure 6.1b, incorporating this array outperforms the base model. While there are no significant differences between models D and E, model D, which does not use auxiliary losses, shows slightly better results.

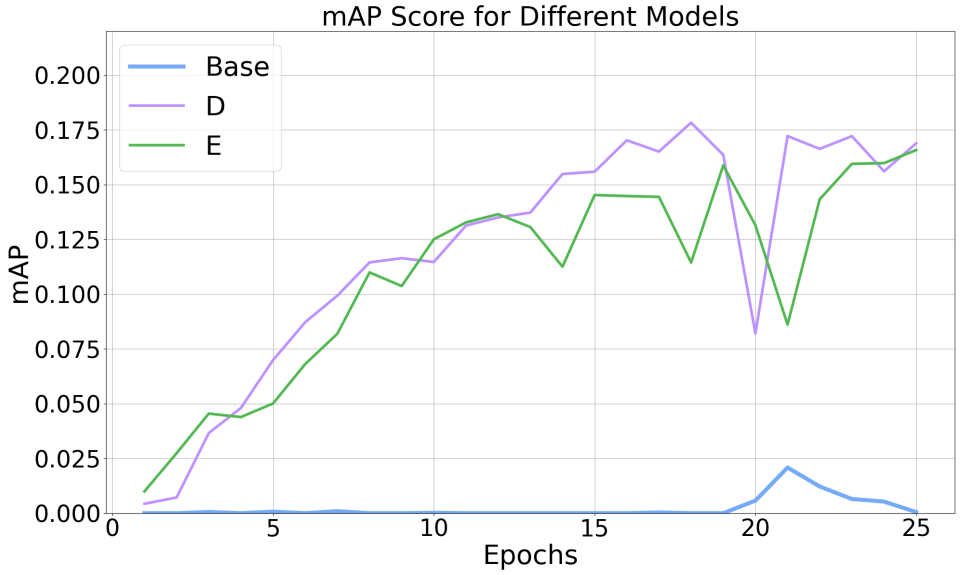
Figure 6.2a presents the third group of different configurations where the focus is on the impact of incorporating 2D predictions into the model. The figure shows that adding 2D predictions to the base model improves performance, as given by model F. Furthermore, incorporating 2D and 3D reference point arrays dramatically improves performance, as seen with models G and H. The previous Figure 6.1b also supports the observation that incorporating reference arrays improves performance. Additionally, model H, which incorporates auxiliary losses, performs slightly better than model G.

The results from the last group are shown in Figure 6.2b. Here it can be seen that incorporating a learnable embedding inspired by the DETR implementation improves performance compared to the base model, although it is still inferior to the other models. Model I and J use one 2D and 3D reference positions from the same learnable array, with model I also adding auxiliary losses. In this case, both models I and J perform well, while model I performs slightly better.

Based on the mAP scores from Figures 6.1 and 6.2, the final model is illustrated in Figure 6.3. This model incorporates auxiliary losses, 2D predictions, a 3D reference point array, a 2D reference point array, and distance weights as in Equation 4.3.

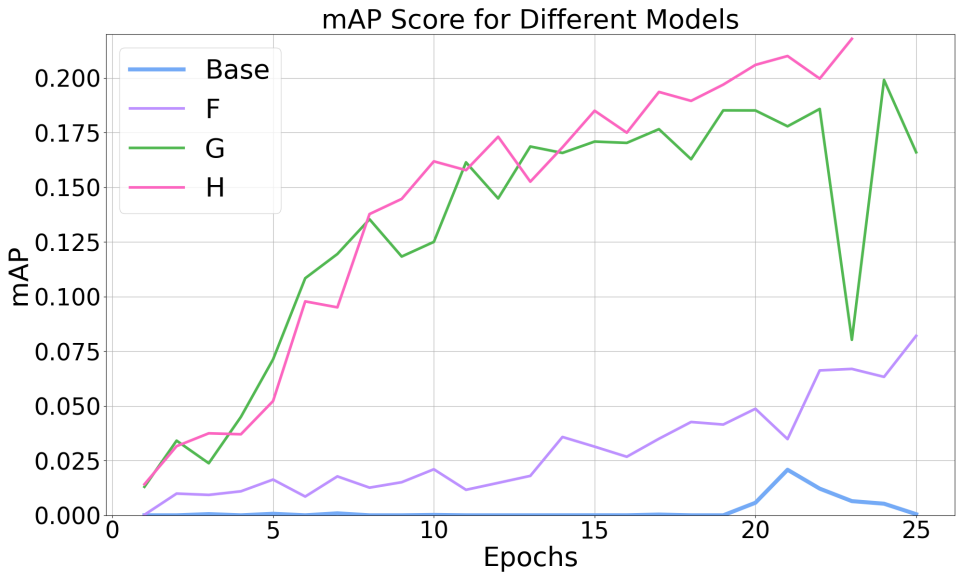


(a) Mean Average Precision (mAP) scores for the first group of different model configurations over 25 epochs. The specific configurations can be found in Table 5.1. The blue line represents the Base model.

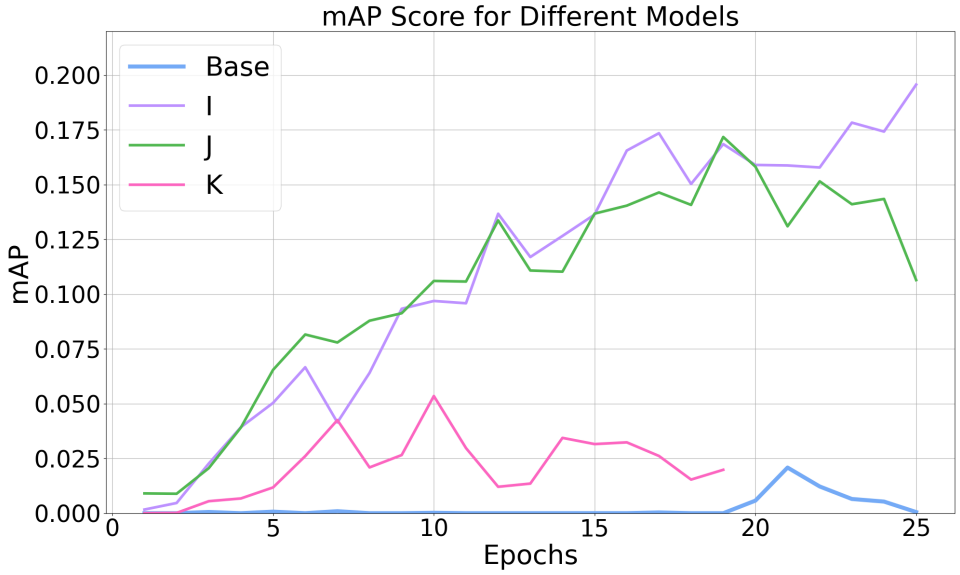


(b) Mean Average Precision (mAP) scores for the second group of different model configurations over 25 epochs. The specific configurations can be found in Table 5.1.

Figure 6.1: Mean Average Precision (mAP) scores for the first and second groups of different model configurations.



(a) Mean Average Precision (mAP) scores for the third group of different model configurations over 25 epochs. The specific configurations can be found in Table 5.1.



(b) Mean Average Precision (mAP) scores for the fourth group of different model configurations over 25 epochs. The specific configurations can be found in Table 5.1.

Figure 6.2: Mean Average Precision (mAP) scores for the third and fourth groups of different model configurations.

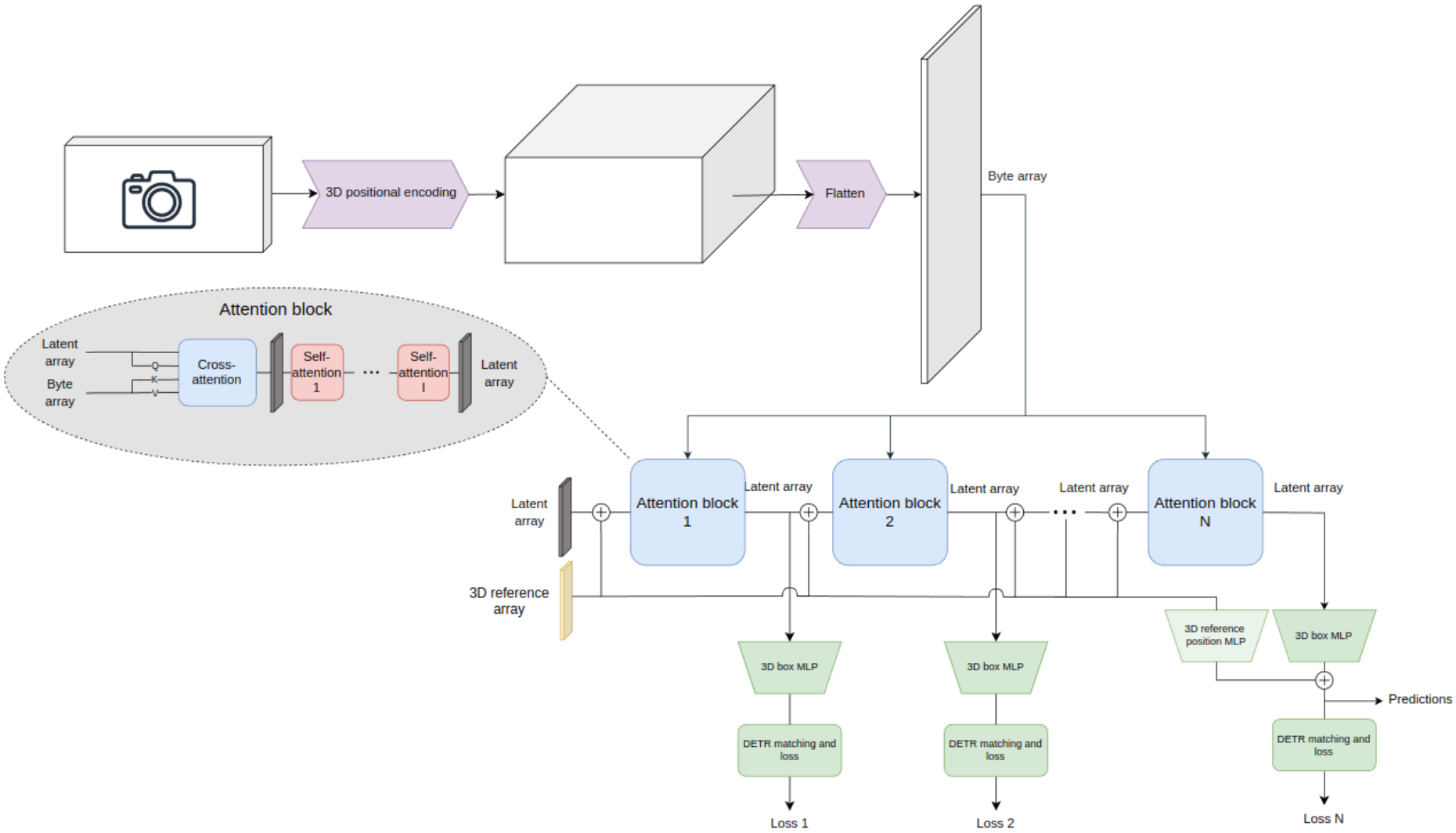
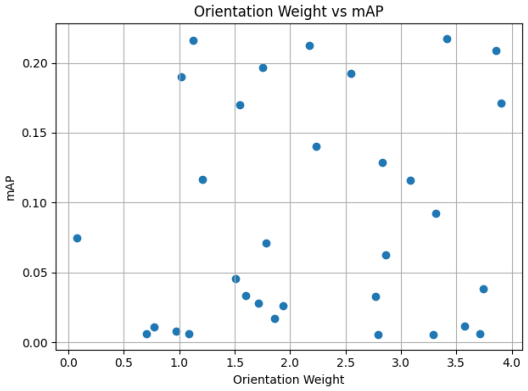


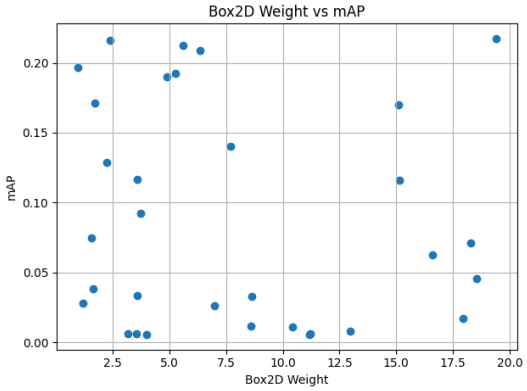
Figure 6.3: Illustration of the model post-ablation study adjustments. N denotes the number of attention blocks, and the first block is explained in detail, where I is the number of self-attentions within each attention block. The latent array inside the attention block is already summed with the 3D reference array. Both N and I are hyperparameters and will therefore be a part of the hyperparameter study.

6.2 Results from the Hyperparameter Study

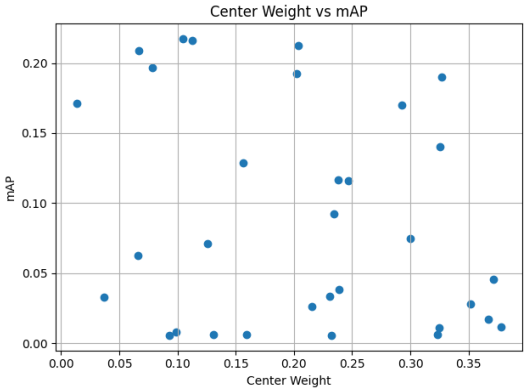
The following Figures show how each hyperparameter affects the mAP score. They show the best value for each hyperparameter and will work as a guide for setting the final values. The following figures show that some plots clearly indicate the optimal values for the hyperparameters, whereas others do not present distinct trends.



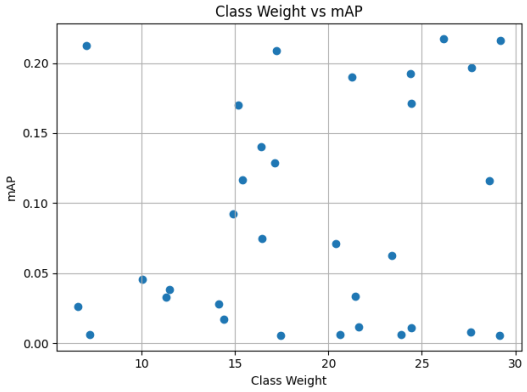
(a) Plot of mAP with respect to the orientation weight.



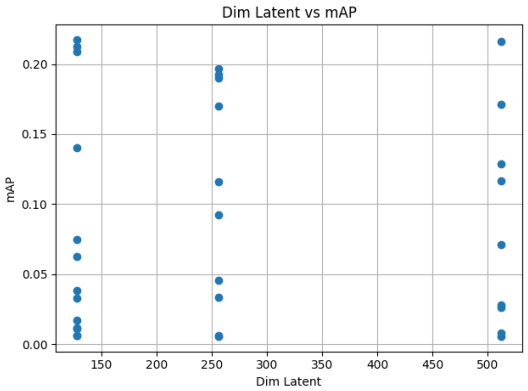
(b) Plot of mAP with respect to the 2D prediction weight.



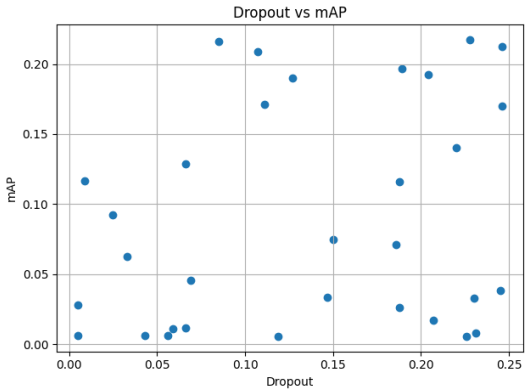
(c) Plot of mAP with respect to the 3D center weight.



(d) Plot of mAP with respect to the class weight.

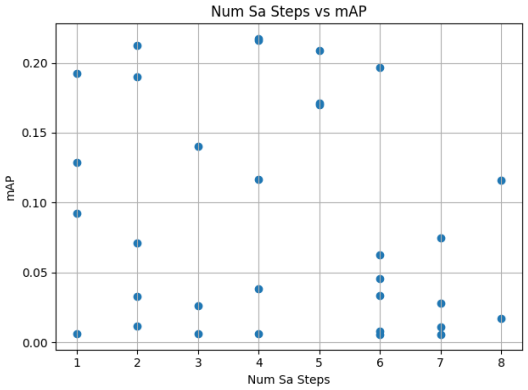


(e) Plot of mAP with respect to the latent dimension.

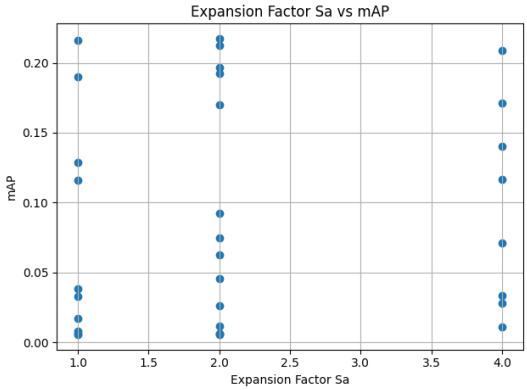


(f) Plot of mAP with respect to dropout rate in Per3D-ObDet.

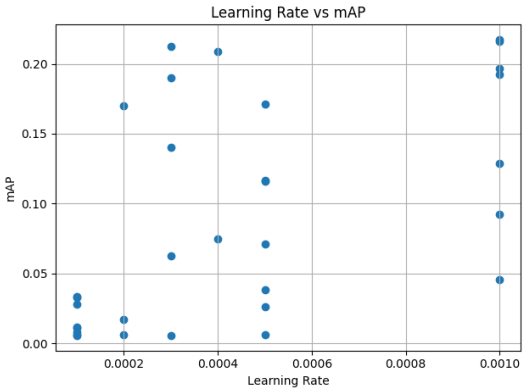
Figure 6.4: Plots of the mAP with respect to six of the hyperparameters within Per3D-ObDet.



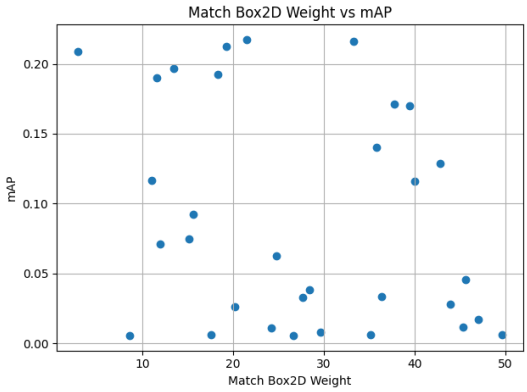
(a) Plot of mAP with respect to the number of self-attention steps within each attention block.



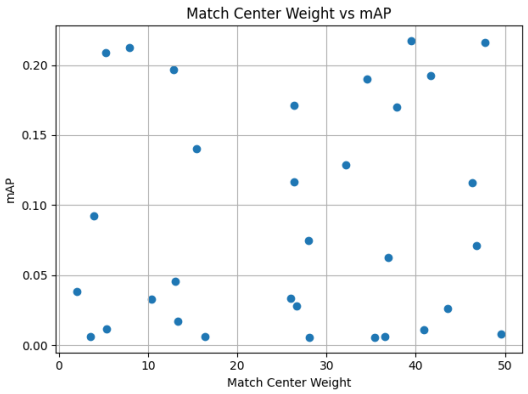
(b) Plot of mAP with respect to the expansion factor in the self-attention step.



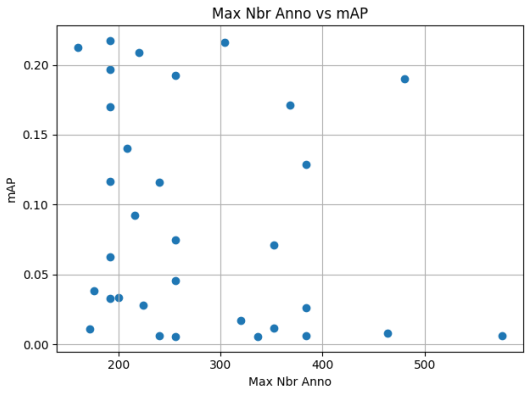
(c) Plot of mAP with respect to the learning rate.



(d) Plot of mAP with respect to the 2D box matching weight.

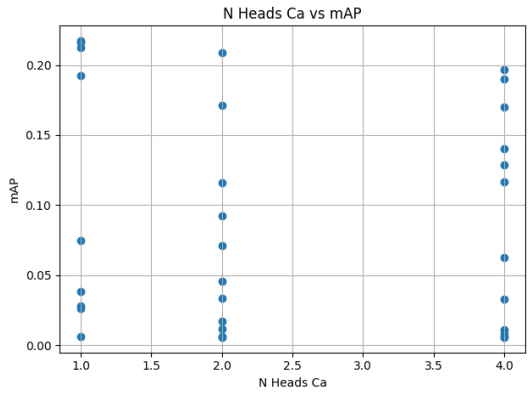


(e) Plot of mAP with respect to the 3D center matching weight.

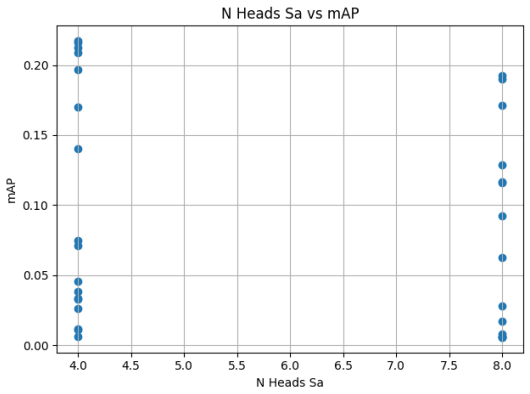


(f) Plot of mAP with respect to the maximum number of annotations for each image.

Figure 6.5: Plots of the mAP with respect to six of the hyperparameters within Per3D-ObDet.

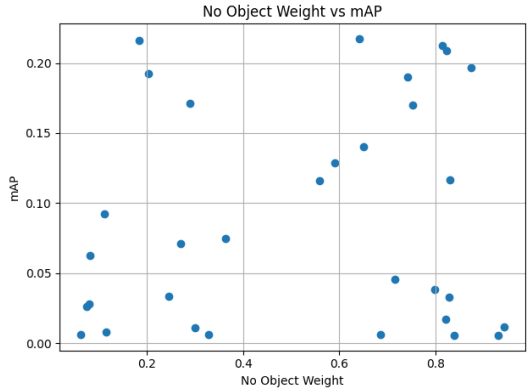


(a) Plot of mAP with respect to the number of cross-attention heads.

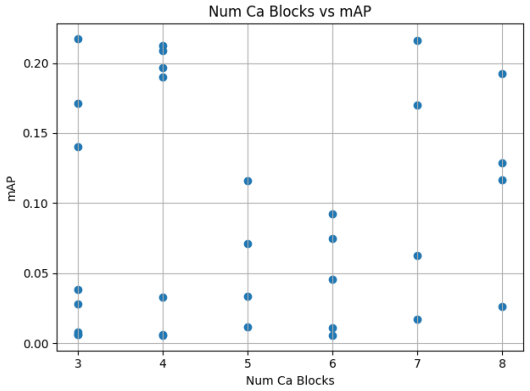


(b) Plot of mAP with respect to the number of self-attention heads.

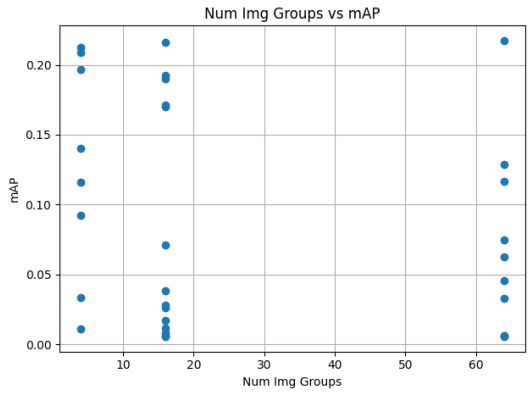
H



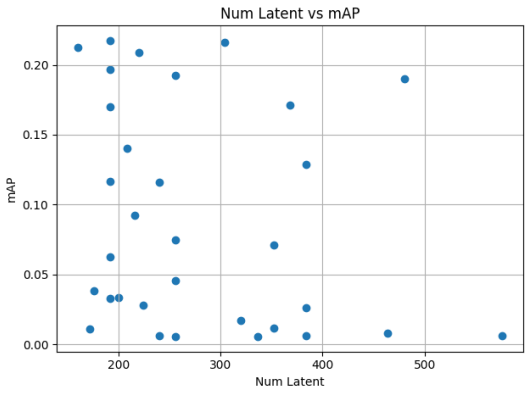
(c) Plot of mAP with respect to the weight for non-objects.



(d) Plot of mAP with respect to the number of attention blocks.

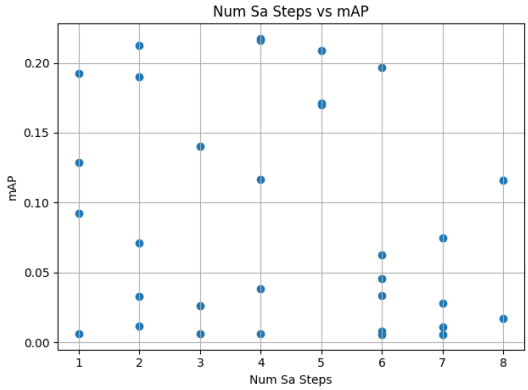


(e) Plot of mAP with respect to the number of cross-attention groups.

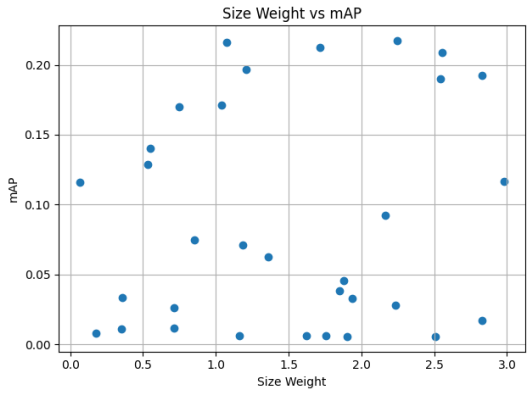


(f) Plot of mAP with respect to the number of latents.

Figure 6.6: Plots of the mAP with respect to six of the hyperparameters within Per3D-ObDet.



(a) Plot of mAP with respect to the number of self-attention steps in each attention block.



(b) Plot of mAP with respect to the size weight.

Figure 6.7: Plots of the mAP with respect to two of the hyperparameters within Per3D-ObDet.

6.3 Final Architecture of Per3D-ObDet

This section presents the final architecture of Per3D-ObDet in detail. Some parts have not yet been stated, and those will be detailed here as well.

Figure 6.9 illustrates the final architecture of Per3D-ObDet. The object detection head MLPs refer to the four MLPs that predict class, bounding box, 2D reference box, and 3D reference center each. The class head consists of two linear layers. The output is then fed through a softmax function to generate class predictions. The bounding box MLP and the 3D reference point MLP consist of three linear layers. Furthermore, the 2D bounding box MLP, as well as the 2D reference position MLP, consists of two linear layers.

Figure 6.8 shows one output from the final model...

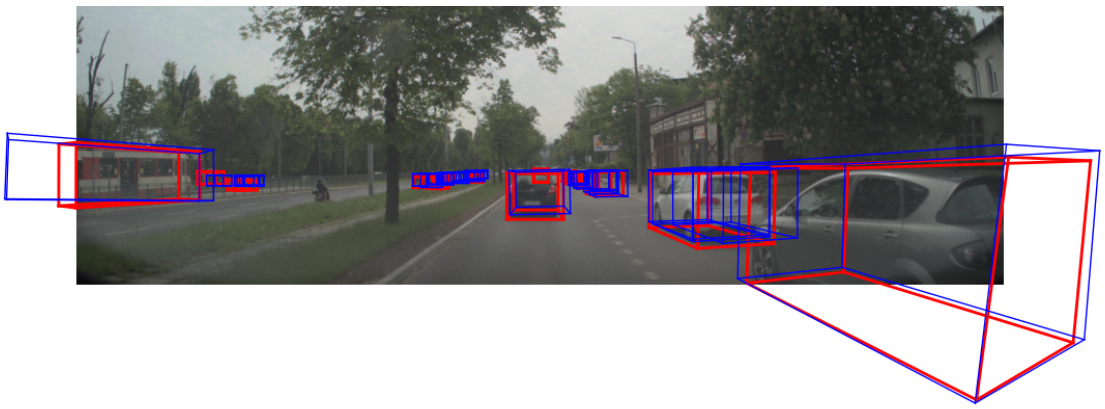


Figure 6.8: An image of the output from Per3D-ObDet. The red boxes are the 3D ground truth, and the blue boxes are predicted 3D bounding boxes.

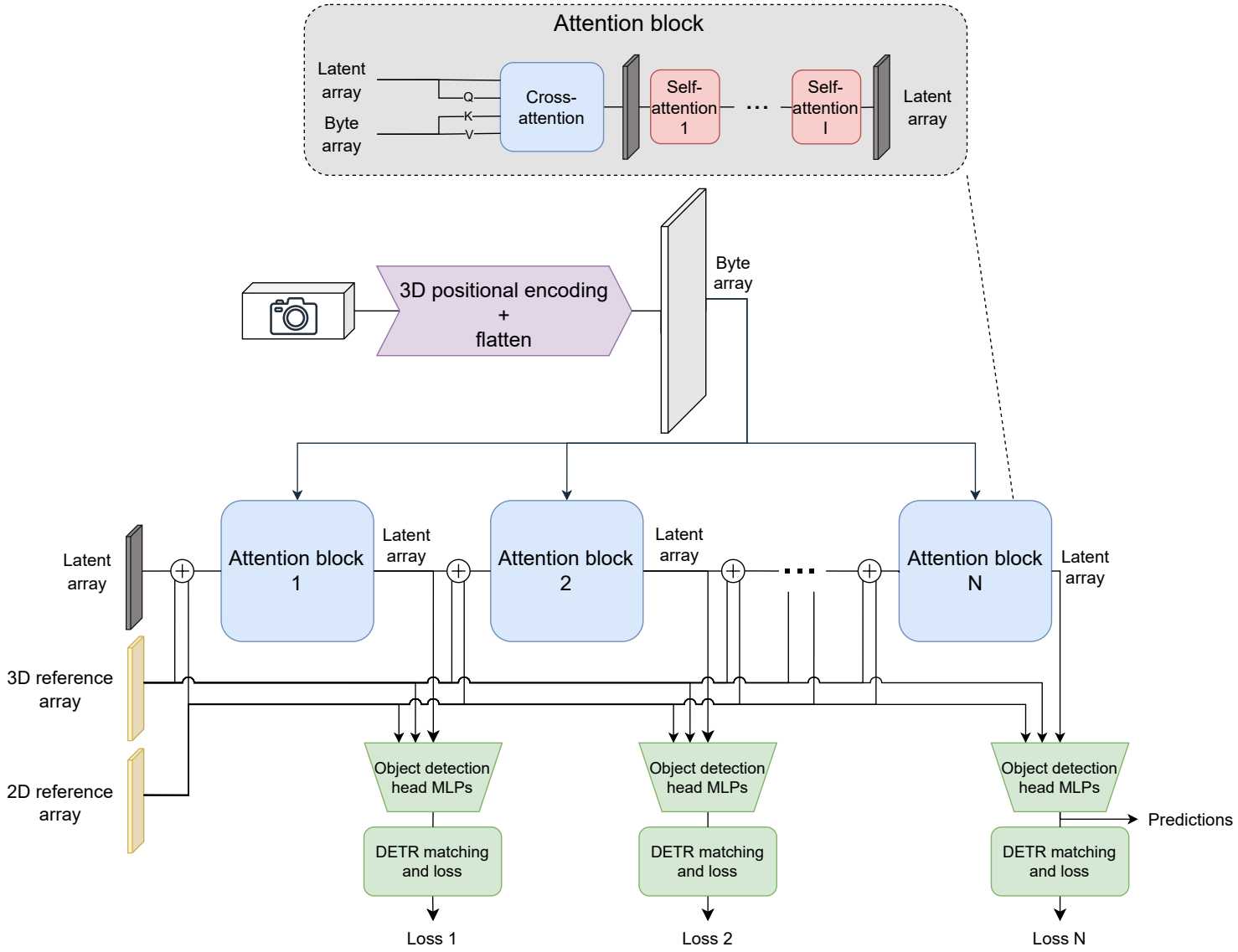


Figure 6.9: Illustration of the final architecture of Per3D-ObDet. The object detection head MLPs refers to the class MLP, bounding box MLP, 2D reference MLP, and the 3D reference MLP...

7
Conclusion

This chapter summarizes the key findings and contributions of this project, as well as the challenges and areas for future improvement. The results from the ablation study have yielded valuable insights into how the original Perceiver architecture must be adapted to allow for 3D object detection. Furthermore, the hyperparameter study provided information on optimizing the model's performance.

7.1 Conclusions and Discussion of Per3D-ObDet

The results show that the Perceiver architecture does not seamlessly adapt for 3D object detection on the provided dataset. Overcoming these obstacles demands extensive modifications and adjustments. While integrating a ResNet backbone led to improvements in certain aspects, it contradicts the inherent flexibility of the Perceiver model, which is designed to handle diverse input modalities seamlessly. Additionally, significant efforts were made to reduce computational complexity to manage the large input sizes. These efforts included scaling down the images during preprocessing and implementing efficient training techniques. Despite these efforts, Per3D-ObDet still requires substantial computational memory and resources, resulting in slow training times.

The ablations study shows that distance weights slightly improve the performance of Per3D-ObDet. However, due to the inherent stochasticity in training models, drawing definitive conclusions from an ablation study based on a single run for each job is challenging. One could expect that distance weights would be more beneficial, as distant objects are typically more challenging to detect with high accuracy. One potential reason for this result could be the selection of the distance-weighting curve. It is possible that the two curves tested in the ablation study, which are quite similar, place too low a weight on some objects, making it difficult for the model to converge. Choosing a curve that is less steep could potentially generate better results.

Moreover, it can be concluded that auxiliary losses improve the performance. This is evident in many of the models, as they frequently perform better than the others in the ablation study. Computing the loss after each attention block forces the model to start making good predictions in the earlier stages of the network.

7.2 Further Improvements of Per3D-ObDet

There are numerous possible avenues for future work in the implementation of Per3D-ObDet. One of the significant challenges encountered in this project is the slow convergence of the matching and loss computation, which is inspired by DETR. Existing research offers various methods to accelerate the convergence of DETR, and some of

these techniques could potentially be applied to improve Per3D-ObDet.

Furthermore, future research can explore different distance weights, given the importance of accurately identifying nearby vehicles in autonomous driving scenarios. Distant objects appear smaller in input camera images, making them inherently harder to identify accurately. This could potentially cause the model not to converge, as noise in the loss on faraway objects might prevent convergence on nearby objects. Ensuring that the model allocates appropriate attention to nearby objects is crucial, as mispredictions for closer objects have more immediate consequences. Hence, having different distance weights could enhance model performance across varying object distances.

One of the key advantages of Perceiver is its ability to handle arbitrary inputs. This arises from the cross-attention between the fixed-size learnable latent array and the input data. This flexibility could allow for easier execution of 3D object detection with other datasets, including those with lower resolutions or easier images for object detection. It also opens up possibilities for sensor fusion in the input data.

Bibliography

[1] The 17 sustainability goals, United Nations. https://sdgs.un.org/2030agenda, Accessed: 2023-12-07.

[2] Mina Alibeigi, William Ljungbergh, Adam Tonderski, Georg Hess, Adam Lilja, Carl Lindstrom, Daria Motorniuk, Junsheng Fu, Jenny Widahl, and Christoffer Petersson. Zenseact open dataset: A large-scale and diverse multimodal dataset for autonomous driving, 2023.

[3] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020.

[5] Rahul Chauhan, Kamal Kumar Ghanshala, and R.C Joshi. Convolutional neural network (cnn) for image detection and recognition. In *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pages 278–282, 2018.

[6] NVIDIA Corporation. How to Train Deep Learning Models on Multiple GPUs. https://doku.lrz.de/files/156140243/155485415/1/1696417441867/data-parallelism-multiple-gpus.pdf, 2023. Accessed: 2024-05-25.

[7] NVIDIA Corporation. Mixed precision training. https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html, 2024. Accessed: 2024-05-29.

[8] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia: case studies on organization and retrieval*, pages 21–49. Springer, 2008.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[10] Alireza Ghasemieh and Rasha Kashef. 3d object detection for autonomous driving: Methods, models, sensors, data, and challenges. *Transportation Engineering*, 8:100115, 2022.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention, 2021.

[14] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, 2021.

[15] Juho Kannala and Sami Brandt. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE transactions on pattern analysis and machine intelligence*, 28:1335–40, 09 2006.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[17] Jayanth Koushik. Understanding convolutional neural networks, 2016.

[18] Yang Li, Si Si, Gang Li, Cho-Jui Hsieh, and Samy Bengio. Learnable fourier features for multi-dimensional spatial positional encoding. *Advances in Neural Information Processing Systems*, 34:15816–15829, 2021.

[19] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[20] Shilong Liu, Feng Li, Hao Zhang, Xiao Yang, Xianbiao Qi, Hang Su, Jun Zhu, and Lei Zhang. Dab-detr: Dynamic anchor boxes are better queries for detr, 2022.

[21] Jiageng Mao, Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. 3d object detection for autonomous driving: A comprehensive survey, 2023.

[22] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018.

[23] Sahil Nokhwal, Priyanka Chilakalapudi, Preeti Donekal, Suman Nokhwal, Saurabh Pahune, and Ankit Chaudhary. Accelerating neural network training: A brief review, 2023.

[24] NVIDIA Corporation. NVIDIA DGX A100 Datasheet, 2020.

[25] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[26] Xin Qian and Diego Klabjan. The impact of the mini-batch size on the variance of gradients in stochastic gradient descent, 2020.

[27] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression, 2019.

[28] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[29] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[31] Yue Wang, Vitor Guizilini, Tianyuan Zhang, Yilun Wang, Hang Zhao, and Justin Solomon. Detr3d: 3d object detection from multi-view images via 3d-to-2d queries, 2021.

[32] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation, 2019.

[33] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M. Ni, and Heung-Yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection, 2022.

A

Appendix 1

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY