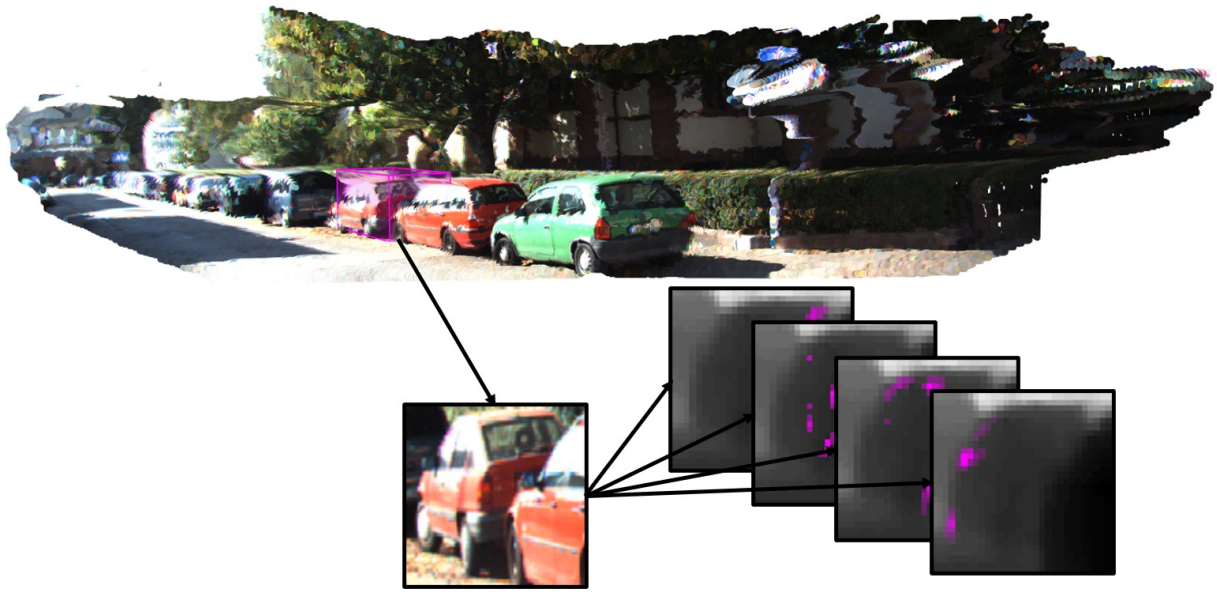
CHALMERS
UNIVERSITY OF TECHNOLOGY



Leveraging Monocular Depth Estimation For 3D Object Detection

Master's thesis in Complex Adaptive Systems

FREDRIK HAGSTRÖM & ARVID WENZEL WARTENBERG

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Leveraging Monocular Depth Estimation For 3D Object Detection

FREDRIK HAGSTRÖM
ARVID WENZEL WARTENBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Master's Thesis 2021
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Leveraging Monocular Depth Estimation For 3D Object Detection
FREDRIK HAGSTRÖM
ARVID WENZEL WARTENBERG

Supervisor: Lennart Svensson, Department of Electrical Engineering
Supervisor: Christoffer Petersson, Zenseact
Supervisor: Adam Tonderski, Zenseact
Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2021
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Pseudo-LiDAR point cloud colored by the corresponding RGB-image from the KITTI [1] data set. The cropped patch RGB channels, and attention maps are illustrated.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Leveraging Monocular Depth Estimation For 3D Object Detection
ARVID WENZEL WARTENBERG
FREDRIK HAGSTRÖM
Department of Electrical Engineering
Chalmers University of Technology

Abstract

3D object detection (3DOD) is a central task in the development of autonomous driving (AD). Camera-based 3DOD methods make use of camera data to produce detections of objects in 3D space. However, a central problem of camera-based 3DOD is that camera data inherently lack depth information, and as such, the performance of these methods has left much to be desired. Recently, deep learning-based algorithms have shown great progress in estimating pixel-wise depth information from monocular RGB-images. Consequently, deep learning-based 3DOD models incorporating these depth estimations have greatly improved the performance of camera-based 3DOD.

The contribution of this work is two-fold. Firstly, we investigate how a state-of-the-art approach for 3DOD based on monocular depth-estimation performs in long-distance settings. We find that the performance improvements from incorporating depth estimations in camera based 3DOD models decrease drastically for very far-away objects. Secondly, we introduce a trainable point selection based on a multi-head attention mechanism. This allows our model to selectively attend to specific regions of the input. We show that the inclusion of our proposed mechanism yields significant performance improvements on Zenseacts data set, especially for problematic settings, such as higher distances. Additionally, we show qualitative results, where the attention mechanism helps the model focus on objects which are heavily occluded.

Keywords: autonomous driving, deep learning, 3D object detection, monocular depth estimation, attention.

Acknowledgements

We would like to thank our academic supervisor Lennart Svensson for his patience and support in discussing and iterating on the ideas which laid the ground for central parts of our work. Furthermore, we would like to thank our supervisors at Zenseact, Adam Tonderski and Christoffer Petersson, for their guidance and openness to provide help with, and discuss, various aspects of the project. Finally, we would like to thank Zenseact for arranging the project, as well as providing the necessary data and computational resources to make it a reality.

Fredrik Hagström & Arvid Wenzel Wartenberg
Göteborg, June 2021

Contents

List of Figures

List of Tables

Acronyms

2DOD 2D Object Detection. 3, 6, 23, 26, 31, 33, 37, 38, 42, 43, 55, 57, 70, 91, 95

3DOD 3D Object Detection. v, xiii, 1–6, 9, 26, 29, 34, 36, 38–40, 62, 70, 87, 92, 95, 97

AD Autonomous Driving. v, 1, 4, 9

ANN Artificial Neural Network. 9

BEV Birds-Eye-View. xv, xvi, 26, 34, 35, 60, 70–81, 89, 92

CNN Convolutional Neural Network. 4, 12, 13, 21, 22, 30–33, 35–37, 44, 46, 66

DORN Deep Ordinal Regression Network. xiii, 30, 31, 35–38, 43, 45, 49

IoU Intesection over Union. xv, xvi, 26, 39, 57, 60, 61, 70–79, 91, 92

LiDAR Light Detection and Ranging. xiii, 1, 2, 4, 29–36, 38–40, 60, 87, 95

LSTM Long Short-Term Memory. 21

MLP Multi-Layer Perceptron. 11–13, 34, 35, 45–47

NLP Natural Language Processing. 16, 20, 21

NMS Non-Maximal Suppression. 32, 34

RCE Relative Center Error. xv, 26, 60, 61, 70–81, 89, 92

ReLU Rectified Linear Unit. xiii, 10, 11, 18

RNN Recurrent Neural Network. 21

ROI Region Of Interest. 43

SGD Stochastic Gradient Descent. 15, 16

1

Introduction

In this chapter, we present a problem background which motivates this thesis. Based on this background, we present our research questions and delimitations.

1.1 Background

The World Health Organization estimates that 1.35 million road fatalities occur every year [3]. Considering that human error is estimated to be a contributing factor to 94% of road accidents [4], research and development of autonomous driving (AD) could save many lives. The Society of Automotive Engineers [5] has defined a scale of varying levels of vehicular autonomy, with the highest level being reserved for fully self-driving vehicles.

In order to achieve the highest level of autonomy, research into AD involves a vast number of components [6]. A basic functionality of the intelligence architecture behind autonomous vehicles is *perception*. Based on data collected via an array of sensors, such as cameras, perception involves processing this data in order to build a representation of the surrounding environment. This representation is then used by other components of the architecture in order to plan the actions of the autonomous vehicle in relation to the world around it. A major element of the perception task is the precise knowledge of the sizes, orientations and locations of surrounding objects, such as other cars or pedestrians, in 3D space. This task is known as 3D object detection.

State-of-the-art 3D object detection methods widely utilize the modern developments in the fields of deep learning and computer vision. Representative examples of the current best-performing 3DOD methods are Frustum PointNets [7], and AVOD [8], which use inputs from image data in combination with LiDAR data. LiDAR data is generated by emitting light beams from a LiDAR unit, the reflections of which are recorded in order to measure the distance to and the locations of surfaces and objects hit by the beam. The resulting data is commonly represented as a 3D point cloud, capturing information about surfaces, and objects in the surrounding environment. While LiDAR data represents a highly accurate view of an environment, it becomes sparser at longer distances and LiDAR units are associated with high costs, amongst other drawbacks [9].

Given the drawbacks associated with LiDAR sensors, there is currently a large incentive for development of 3DOD models which rely more exclusively on camera data. A main problem with purely camera-based methods in comparison with LiDAR-based ones is that camera data contains no depth information, which greatly increases the complexity of the camera-based 3DOD task. Recently, image-based depth estimators [10, 11, 12, 13] have been leveraged in camera-based 3DOD pipelines in order to account for the lack of depth information [14, 15, 16, 17, 18, 19, 20]. DORN [12] is an example of a commonly used a neural network-based depth estimation method, which takes as input monocular image data and outputs the estimated depth value of each pixel in the image. These so-called *mono-depth* estimations provide the camera-based methods with the additional depth information of the environment. However, many mono-depth-based 3DOD pipelines still perform poorly in comparison to LiDAR methods.

In [18], it is concluded that a key factor of the performance gap between LiDAR-based methods and mono-depth-based methods is the choice of data representation. By transforming the depth maps to a 3D point cloud representation, called Pseudo-LiDAR, one can make use of state-of-the-art LiDAR pipelines which take point clouds as input. The performance over previous camera-based models is significantly increased, and the Pseudo-LiDAR point cloud is concluded to be a key component in this success. However, in [19], these conclusions are more carefully investigated and it is shown that the success of Pseudo-LiDAR is not due to the Pseudo-LiDAR data representation in and of itself, but rather from the coordinate transform used to unproject the pixel-wise depth estimates to the Pseudo-LiDAR representation. Since the point cloud representation is shown to be of less importance than previously assumed, Pseudo-LiDAR-based methods are no longer bound to the point-wise methods otherwise assumed for point cloud data, and more powerful convolutional backbone networks may be employed instead. A convolutional architecture named PatchNet is subsequently proposed, achieving a new state-of-the-art on the KITTI [1] data set.

In this thesis, we investigate the performance of PatchNet [19] on a data set provided by Zenseact. This data set contains objects at much longer distances than the commonly used KITTI [1] benchmarking data set. We expand on previous work by investigating how the performance of PatchNet scales with increased distances. Given the introduction of convolutional neural networks in PatchNet, we also investigate how additional input channels, such as RGB-data, can enhance 3DOD performance in combination with mono-depth estimations. Finally, we highlight drawbacks with the binary mask utilized in PatchNet, introducing an attention mechanism to resolve some of the observed issues. With our proposed mechanism, we achieve significant performance improvements over PatchNet, especially for higher distances.

1.2 Problem Statement

This section presents our research questions. We first give an overview of the specific architecture which is most central to this work, such that the reader can grasp the

context of our research questions.

At the highest level of abstraction, the problem of monocular image-based 3DOD, which is studied in this thesis, is illustrated in Figure 1.1.

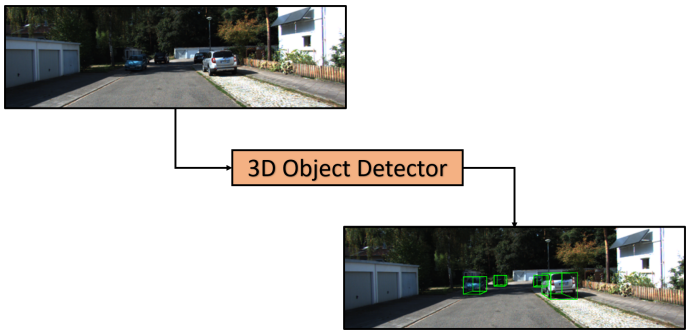


Figure 1.1: High-level overview of the 3DOD problem. Here, the 3D bounding boxes are illustrated as green boxes in the image.

The above figure illustrates that monocular image-based 3DOD amounts to building a 3D Object Detector, which, given an input RGB-image, produces a set of 3D bounding boxes for the detected objects contained in the image.

In this thesis, we perform additional intermediate tasks, leveraging mono-depth estimation and 2D object detection networks for the 3DOD problem. Figure 1.2 illustrates these additional tasks.

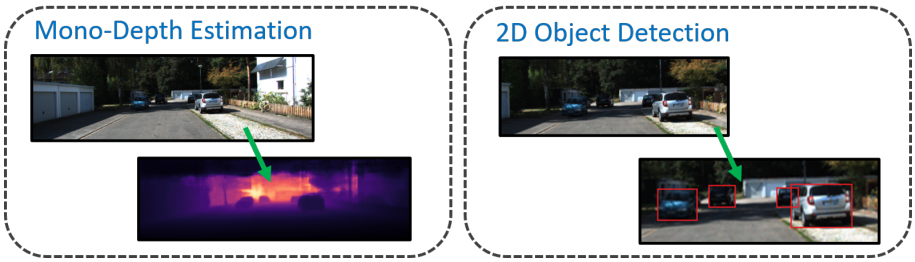


Figure 1.2: The Mono-Depth Estimation step takes a monocular image as input, and produces pixel-wise estimates of the physical depth values in the image. The 2D Object Detection step detects objects in the image, producing a 2D bounding box for each detected object.

The specific 3DOD pipeline we expand upon in this work is PatchNet [19]. PatchNet utilizes the additional mono-depth and 2DOD information for the 3DOD task. The central tasks performed in PatchNet which utilize this additional information are illustrated in Figure 1.3.

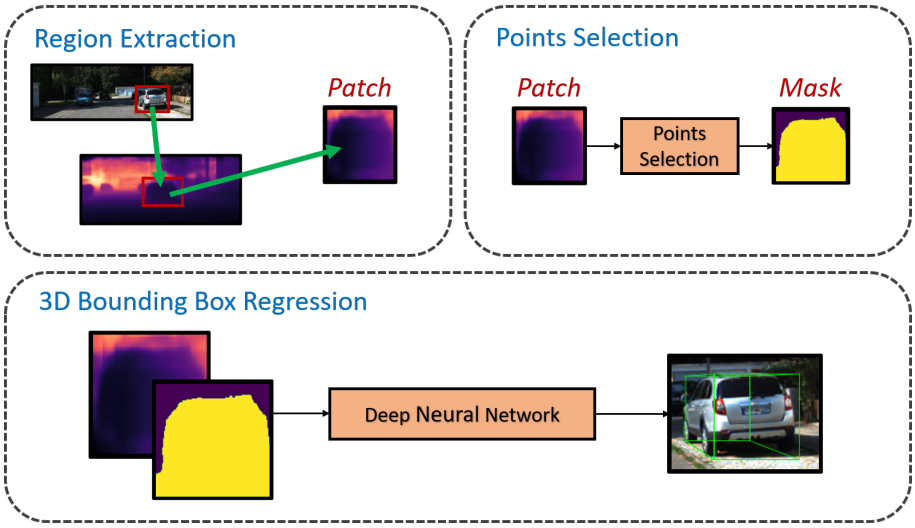


Figure 1.3: Tasks performed in PatchNet to leverage mono-depth and 2D detections for the 3DOD task.

In the Region Extraction task, PatchNet takes a 2D detection made by the 2D detector, and extracts the corresponding region from the pixel-wise depth estimate, producing a so called *Patch*. In the Points Selection task, PatchNet produces a *mask* using a heuristic approach, aimed at selecting points in the Patch which belong to the detected object. Finally, for each Patch, a deep neural network takes the Patch and Mask as input, outputting a final 3D bounding box. Since each 2D detection yields exactly one patch, the intermediate tasks are repeated for each 2D detection in the image to produce the final set of 3D bounding boxes for a given image.

In the literature, 3DOD models in the field of AD (including PatchNet) are very commonly evaluated on the KITTI [1] benchmarking data set. However, the KITTI data set only contains objects up to approximately 75 meters, whereas the data set provided by Zenseact contains objects at much greater distances. Since the error of image-based depth estimates has strong scaling with distance [18], our first of research question relates to evaluating PatchNet in different distance settings:

How does the mono-depth based 3DOD pipeline PatchNet perform in settings of increasing distance?

As stated in Section 1.1, it is posited in [19] that the success of Pseudo-LiDAR based methods is not due to the point cloud representation itself, but rather the coordinate transformation used to unproject the pixel-wise depth estimates to the Pseudo-LiDAR representation. This allows for an image representation of Pseudo-LiDAR data, enabling the use of the more general and powerful CNNs. Since CNNs are traditionally used to process other types of channels, such as RGB image data, it is natural to expand upon the experiments performed in [19], and explore the effect of incorporating additional input channels to the PatchNet pipeline. As such, we formulate our second research question as:

Can additional input channels, such as RGB channels, improve the performance of

PatchNet?

In PatchNet, a points selection process is introduced, based around a binary mask, with the aim of masking out irrelevant background points. The mask works by discarding all pixels with a depth value exceeding the mean depth value plus a small offset value, and keeping the rest. The information about which pixels too keep and toss is intermediately stored in a matrix we refer to as the *binary mask*,

$$M(u,v) = \begin{cases} 0, \text{if pixel } (u,v) \text{ is discarded} \\ 1, \text{if pixel } (u,v) \text{ is kept.} \end{cases}$$

In [19], this is likened to a hard attention mechanism, which forces attention on pixels belonging to the foreground. The binary mask performs poorly in certain situations, such as for occluded objects. Figure 1.4 illustrates a failure-case of the binary mask, where an occluded object is masked out.

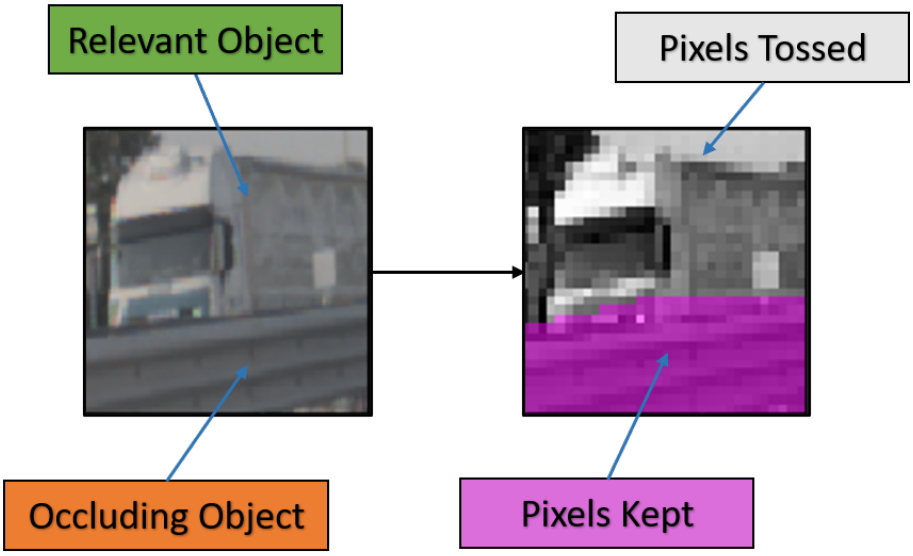


Figure 1.4: Illustration of information loss from using the PatchNet binary mask. The left image shows an occluded object, and the right image shows, in pink, which pixels are selected as relevant.

The pink regions corresponds to the pixels which are kept for further processing in the pipeline, whereas all other pixels are discarded. As such, the above figure shows a majority of the pixels belonging to the relevant object are discarded, whereas none of the irrelevant foreground pixels are excluded. In order to address this problem, we replace the binary mask with an attention mechanism, allowing the model itself to selectively focus on specific regions of an image in order to mitigate cases such as the above. Formally, our final set of research questions is:

Can the 3DOD performance of PatchNet [19] be improved, by means of replacing the binary mask with an attention mechanism? What are the strengths and weaknesses of this mechanism?

1.3 Delimitations

In this work, we treat the 2DOD and Mono-Depth Estimation steps as-is. In other words, we do not optimize an end-to-end pipeline with respect to these networks. This is very common in literature [19, 7, 20], and as a result, we treat the outputs of those networks as a fixed data set.

Given the complexity of the problem at hand, we are faced with a very large hyperparameter space. As such, we shall not make any claims of optimizing the hyperparameters to any great extent.

Annotated data is provided by Zenseact. Due to the confidential nature of our agreement with Zenseact, we will be somewhat limited in describing exact details of the data. Additionally, we will utilize 2DOD and monocular depth estimation models developed by Zenseact, leading to similar restrictions in detailed descriptions.

The specific subset of Zenseact data to which we have access does not contain any test set employed for the Baseline network. Since confidentiality agreements constrain us to reporting performance relative to the baseline, one delimitation of this work is that we evaluate the performance of our models solely on the validation set. As such, we do not make claims about how well our models would perform if they were deployed. Instead, our results on the validation set are analyzed, and as such, should be seen as comparative results between the isolated architectural modifications explored in this work.

Finally, this work largely ignores the importance of computational efficiency. Instead, our focus is almost purely on the sheer performance of the studied models, regardless of the time-complexity of processing data. However, it is important to keep in mind that performing such an analysis is crucial when mapping out application prospects for real-time autonomous driving.

1.4 Contributions

Contrary to the commonly used KITTI [1] benchmarking data set which contains objects at up to approximately 75 meters, Zenseacts data set has plenty of objects at much greater distances. We use this data to expand on the work done in the literature, investigating how the state-of-the-art PatchNet model for 3DOD based on monocular depth-estimation performs in the previously unseen long-distance setting. We find that the performance improvements from incorporating depth estimations in camera based 3DOD models which is reported in the literature decrease drastically for very faraway objects. Additionally, we find negative effects on the overall performance of the network when including faraway objects in the training data set, as well as a lack of generalization ability across distances.

We highlight failure cases of the binary mask utilized in PatchNet (see Figure 1.4). Model improvements are proposed by replacing the heuristic method with an attention mechanism, which also utilizes additional RGB input data. With these changes,

considerable performance improvements over PatchNet are observed on Zenseacts data set, especially for problematic settings such as higher distances. Finally, we show qualitative results, illustrating how the attention mechanism consistently attends to meaningful areas of an input image, which helps identify heavily occluded objects.

2

Perception In Autonomous Driving

In this chapter, we cover fundamental aspects of deep learning, and move on to describe specific applications of the theory to 3DOD within AD which are relevant to this thesis.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are a class of computing systems which are mainly used for machine learning. ANNs were originally conceived with the human brain in mind, and as such, they are often simply referred to as *neural networks*. At a high level, ANNs consist of a number of computational units, so-called *perceptrons*, which are organized in layers and linked together. Each unit computes a function of the weighted sum of its inputs, and as such, data can be introduced at the input layer and then fed forward through all units to the output layer, to produce an output. This input-output relationship can be used to model functional relationships between input data and desired target outputs, and as the networks grow more complex, the complexity of the functions they are capable of modeling increases accordingly.

2.1.1 Perceptrons

Inspired by the human brain, where neurons are the fundamental units of computation, the idea of *perceptrons* can be traced back to McCulloch and Pitts [21]. Perceptrons constitute the basic building block of an ANN. Figure 2.1 illustrates the modern perceptron.

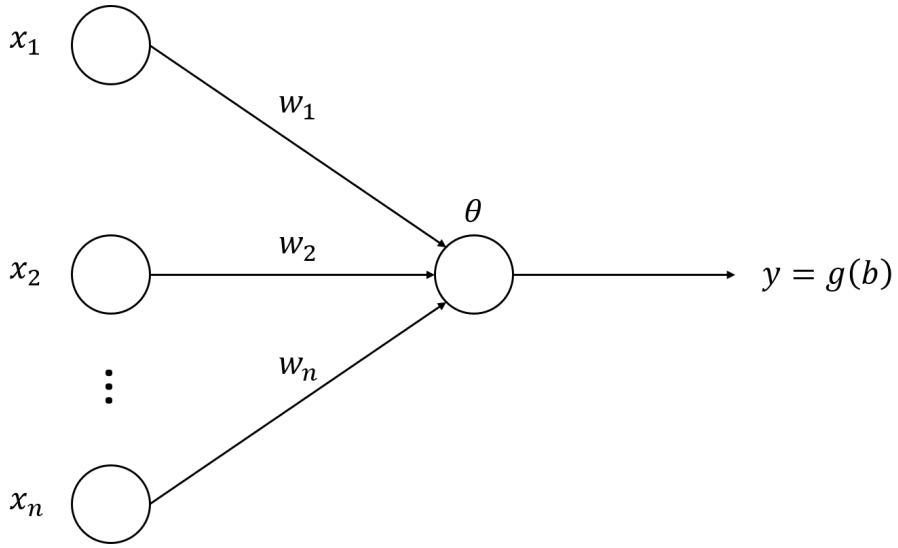


Figure 2.1: Schematic illustration of a perceptron.

In the figure, we see that the final output of the perceptron is

$$y = g\left(\mathbf{w}^T\mathbf{x} - \theta\right) \tag{2.1}$$

Receiving a vector inputs $x_i, \quad i = 1, ..., n$, weighted by w_i, we see in the figure that the perceptron outputs $g(b)$. Here $b = \mathbf{w}^T\mathbf{x} - \theta$ is the *local field*, θ is the the threshold of the neuron, and g is the *activation function*. Activation functions also serve the purpose of injecting non-linearity into the networks, which is a crucial component to the success of neural networks.

Typical choices for the activation function g are the Rectified Linear Unit (ReLU), the Sigmoid function σ, and the tanh function. They are given mathematically by

$$\begin{aligned} \mathrm{ReLU}(x) &= \max(0, x), \\ \sigma(x) &= \frac{1}{1 + \mathrm{e}^{-x}}, \\ \tanh(x) &= \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}}, \end{aligned}$$

and are depicted in Figure 2.2.

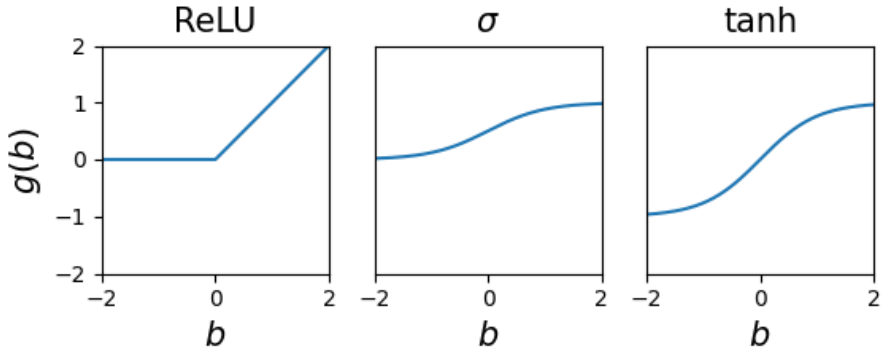


Figure 2.2: Plots of the ReLU, Sigmoid and tanh functions.

Each activation function in Figure 2.2 has its own advantages and use-cases. For example, the ReLU activation may help counteract the effects of the vanishing gradients problem [22] (see section 2.2.2), while the Sigmoid function is useful for modelling probabilities, as it is constrained to the unit interval, and can therefore be used for classification tasks. The tanh-function constraints the outputs to the interval $(-1, 1)$.

The Sigmoid function also has a multivariate analogue, the Softmax function, defined as

$$\text{Softmax}(\mathbf{x})_i = \frac{\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{x_j}}.$$

By construction, the Softmax function maps a vector of outputs $\mathbf{x}$ to another positive vector with unit sum. When applied to the outputs of a neural network, each output unit can therefore be thought of as a class probability vector. As such, the Softmax function is commonly used for multi-class classification problems.

2.1.2 Multi Layer Perceptrons

While the perceptrons discussed in the previous section are designed to reflect the computations of a neuron in the human brain, a single perceptron is only capable of very simple tasks. In a multi-layer perceptron (MLP), layers of perceptrons are chained together sequentially, illustrated in Figure 2.3.

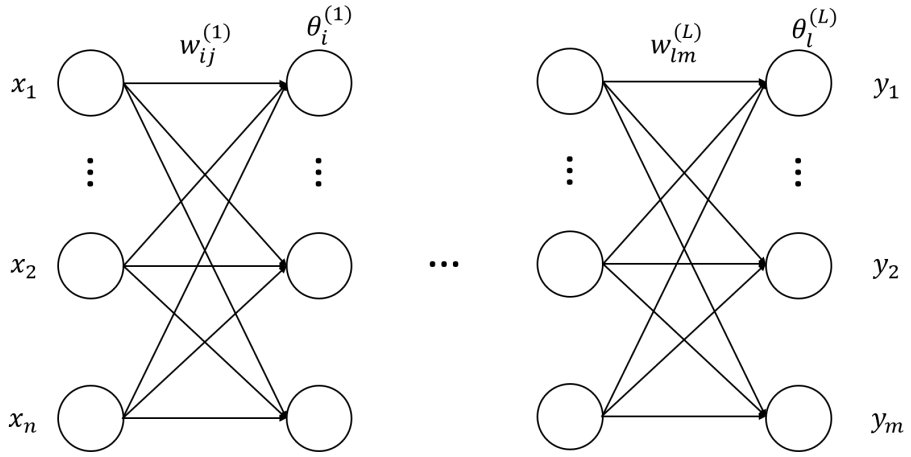


Figure 2.3: Illustration of a Multi Layer Perceptron.

In the above illustration, the perceptrons in each layer receive inputs from the perceptrons in the previous layer, starting from the input layer x_i. For example, the output of the ith neuron in the first layer is $y_i^{(1)} = g(\sum_j w_{ij}^{(1)} x_j - \theta_i^{(1)})$. The output of the MLP is quantified by the activations y_i of the perceptrons in the final layer. Layers in between the input- and output layer are called *hidden layers*. It is useful to interpret the MLP as a function $\mathbf{y} = F_\theta(\mathbf{x})$ which maps the input vector $\mathbf{x}$ to the output vector $\mathbf{y}$, such that

$$F_\theta : \mathbf{R}^n \longrightarrow \mathbf{R}^m, \quad \mathbf{x} \in \mathbf{R}^n, \mathbf{y} \in \mathbf{R}^m$$

and where θ represents the parameters of the model, which are the weights and biases of the perceptrons which make up the MLP. Due to the non-linear activation functions commonly used in MLPs, functions implemented by MLPs with multiple hidden layers become capable of representing an immensely complex family of functions. The exact layout of the network can vary drastically depending on the application, and is commonly referred to as the network *architecture*

2.1.3 Convolutional Neural Networks

As the size of input data increases, the use of MLPs becomes less and less feasible due to how the numbers of required parameters scales with the increased input size. A very common example of this is image data, which can often contain millions of pixels. Such an image would require millions of parameters for each unit in an MLP, as each unit is usually connected to all units in the previous layer. To this end, it has become very common to use Convolutional Neural Networks (CNNs) when dealing with image data [23]. CNNs represent each neuron as a kernel that slides across an input image, as illustrated in Figure 2.4, computing a transformed representation similarly to a perceptron. Since the kernel only operates on local neighbourhoods of the input, the number of parameters in the model can be drastically reduced as compared to an MLP.

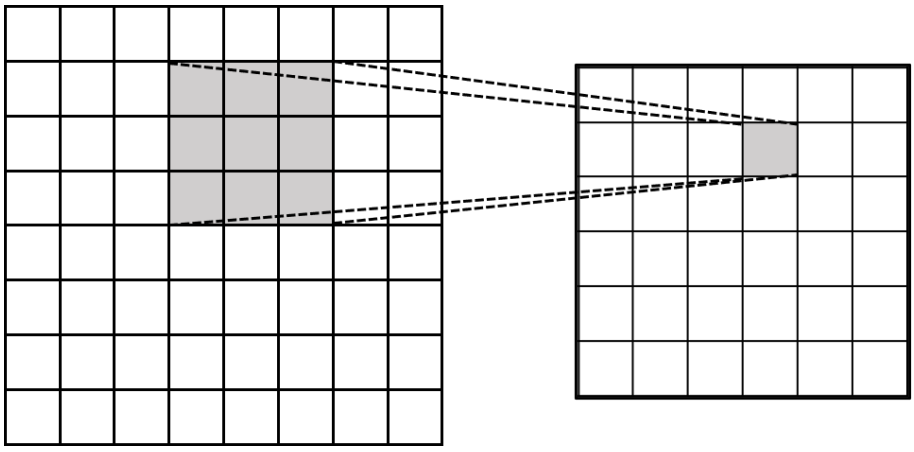


Figure 2.4: 2D convolution with a 3×3 kernel. An input patch is transformed via the convolution operator to an output *feature map*.

The process of sliding the kernel across an input channel and calculating the output results in a *feature map* (right in the above figure). In the above figure, we see how exactly one value in the output feature map is produced. Specifically, the grey-marked squares in the left represent inputs $\mathbf{x}$, and the value of the gray square on the right is computed by use of Equation 2.1, using the kernel weights $\mathbf{w}$ and bias θ. This calculation process is repeated at each position to produce a complete feature map. In addition to the reduced number of required weights required by a CNN compared to an MLP, the nature of the convolution operation also captures local dependencies between pixels and makes the kernels translation invariant. This is especially useful for image data, since images can contain objects and distinguishing features at any position.

Much like MLPs contain chained layers of perceptrons, CNNs usually consist of a large number of chained *convolutional layers*, each of which consists of n kernels. As such, each feature map in a CNN layer can be thought of as analogous to one unit in an MLP layer. In the case of image data, the data is usually three dimensional with dimensionality $C \times H \times W$, where C is the number of input channels ($C = 3$ in the case of RGB-data), and H and W are the spatial dimensions (representing the pixel positions of the image). Given an input with dimensionality $C \times H_{\text{in}} \times W_{\text{in}}$, it acquires dimensionality $n \times H_{\text{out}} \times W_{\text{out}}$ after being fed through a convolutional layer with n kernels. Note that the spatial dimensions $(H_{\text{in}}, W_{\text{in}})$ and $(H_{\text{out}}, W_{\text{out}})$ need not necessarily be equal, since the kernel cannot be applied to the edges of the input map when the kernel size is greater than 1×1. In some cases, one requires input and output shapes to be equal, which is usually achieved by applying a *padding*, whereby pixels (usually of value 0) are appended to the edges of the image/feature map.

In some cases, it is desirable for the output feature maps to have smaller spatial dimensions than the input maps, which is achieved via *downsampling* of the feature map. A common way of downsampling is the use of *pooling layers*. A pooling layer works much in the same way as a regular convolutional layer, but instead of

computing the kernel function at each position of the input map, a deterministic function is used. Common examples of such functions are the average and maximum functions, resulting in an *average pooling layer* and *max pooling layer*, respectively. As such, the operation is identical to the operation shown in Figure 2.4, with the only difference being that instead of computing the transformation implemented by the kernel parameters, the function applied to the input patch is the average function or max function, respectively.

2.1.4 Training Neural Networks

The process of optimizing neural networks for a task is most commonly referred to as *training*. The type of machine learning task influences not only network architecture, but also the type of training process. Machine learning tasks are usually divided into two categories [23]:

- **Supervised learning**: Given a data set consisting of input-output pairs, the model is tasked with finding the functional relationship between inputs and outputs. In this context, this function can be parametrized by a neural network. The term "supervision" then refers to the fact that the model is "supervised" during training by being fed the desired labels for each input example. Examples of supervised learning are regression, where a value is predicted for a given input, and classification, where an input is classified into a category.

- **Unsupervised learning**: Unsupervised learning concerns those tasks where there are no predefined labels to which are inputs are to be mapped. Instead, the task is to find patterns and heuristics which describe the data, without the use of pre-defined labels. Examples include cluster analysis where data is separated into clusters depending on different measurements of similarity, and anomaly detection, where anomalous data points are detected.

Though there are additional learning modes (e.g., self-supervised learning, semi-supervised learning, reinforcement learning), the main focus of this report is supervised learning.

More formally, given a *data set* of observations $\mathcal{O}$ of input-target pairs $(\mathbf{x}_\mu, \mathbf{y}_\mu) \in \mathcal{O}$ sampled from some function F, the task of supervised learning involves finding a model $\hat{\mathbf{y}} = F_{\boldsymbol{\varphi}}(\mathbf{x}, \hat{\boldsymbol{\varphi}})$ such that when given an input $\mathbf{x}_\mu \in \mathcal{O}$, it produces the correct output $\hat{\mathbf{y}}_\mu = \mathbf{y}_\mu$. In order to find such a function, one can optimize the parameters of a neural network with respect to a *loss function* $\mathcal{L}$, which gives some measure of the error between the output of the network and the true target value. The loss function is a smooth function that obtains its global minimum when the model reproduces the correct output for each input from the set of observations. The loss function is thus a function of the set of observations, and the model parameters:

$$\mathcal{L} := \mathcal{L}(\mathcal{O}, \boldsymbol{\varphi})$$

The choice of loss function is crucial for the network to succeed at a desired task. In

continuous regression tasks, it is common to use the mean squared error, defined as

$$\mathcal{L} := \frac{1}{n} \sum_{\mu=1}^{n} \sum_{i} (y_{\mu,i} - \hat{y}_{\mu,i})^2$$

where the outer sum is taken over the examples of the data set, and the inner sum is taken across the output dimension. When this function reaches its global minimum, then the model output is equal to the corresponding target value for all data points. Additionally, optimization of the mean squared error puts no constraints on the values of the model output, which makes it especially suitable for regression. For classification tasks, the cross-entropy loss is often used, defined as

$$\mathcal{L} := -\frac{1}{n} \sum_{\mu=1}^{n} \sum_{i} y_{\mu,i} \log \hat{y}_{\mu,i}.$$

When the outputs of the model are restricted to the unit interval via the Sigmoid or Softmax activation function they can be interpreted as probabilities. By minimizing the cross-entropy loss, the outputs will correspond to the true class probabilities for the input.

2.1.5 Stochastic Gradient Descent

Given a network architecture and a loss function, the training of the network involves tuning the network parameters with respect to the loss function, such that the loss function is minimized. *Gradient descent* is a classical optimization algorithm, whereby the parameters $\boldsymbol{\varphi}$ of a model are updated iteratively. During each iteration, the parameters are updated using update rule given by

$$\boldsymbol{\varphi} \longrightarrow \boldsymbol{\varphi} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\varphi}}. \tag{2.2}$$

Note that it is now useful to view the loss function as a function of the model parameters rather than the output-target pairs, and the gradient of the loss function is computed with respect to each parameter. This update rule can be interpreted as taking a step of length proportional to the *learning rate* η, along the direction of the steepest gradient in the loss "landscape". The computational algorithm for calculating the gradient of the loss function with respect to the model parameters in the case of neural network training is based on the chain rule and known as *backpropagation* [24].

Classical gradient descent is often not feasible in cases where the data set is very large, as is often the case in modern machine learning applications. To this end, *stochastic gradient descent* (SGD) is commonly employed, especially for the training of neural networks [25]. Instead of calculating the loss function with respect the entire data set at once, the data set is randomly split into a set of *mini-batches* $\mathcal{B}_i \subset \mathcal{O}$ of equal size, i.e., subsets of the original data set. The loss function is the calculated with respect to one such mini-batch at a time, and the model parameters are updated using equation 2.2 based on this batch. Once each mini-batch has been

processed, we have completed one *epoch*, i.e., one pass over the entire data set. The training of a neural network commonly involves several such epochs, such that each data point is used for parameter updates more than once.

In recent years, a number of extensions of SGD have been proposed, such as AdaGrad [26] and and Adam [27]. The common theme amongst these extensions is that they include some degree of adaptation of gradients and learning rate, in order to mitigate issues with the regular SGD algorithm. These algorithms are very commonly used in practice today.

2.2 Deep Learning

In 2012, AlexNet [28] revolutionized the field of computer vision by achieving a drastically improved state of the art performance on the image recognition benchmark ImageNet [29]. AlexNet is a deep convolutional network, meaning that it is a neural network consisting of a large number of stacked convolutional layers. Since AlexNet, image recognition benchmarks have consistently seen deep convolutional neural networks reach the highest performance. Along with advancements in hardware, research into deeper neural networks has surged, and AlexNet is now widely seen as an early milestone in the modern field of *deep learning*.

Deep learning is the study of deep neural networks, i.e., neural networks with a large number of hidden layers. Due to their non-linearity and sheer number of parameters, deep neural networks have an immense representation capacity, and often succeed at tasks where simpler machine learning algorithms, such as shallow neural networks, fail. Deep learning has been successfully applied in many different fields, such as computer vision [28], Natural Language Processing [30] (NLP) and strategic games such as go [31] and chess [32].

2.2.1 Training a Deep Neural Network

The ever increasing depth of deep neural networks also comes with a number of disadvantages. First and foremost, given the amount of parameters involved, they require very large data sets to be trained. Modern deep learning research can involve networks with parameter counts in the billions and data sets on the order of magnitude of the content of the entire internet [30]. As the number of parameters increases, the networks are prone to *overfitting* if the data set is too small. Overfitting is the phenomenon where the model has a capacity much greater than what the data set can support, which causes the network to easily over-specializes on the training set, at the cost of generalization ability. In other words, the network learns the training set exactly, but fails when confronted with an unseen sample from a validation set.

To combat the problem of overfitting, one usually employs different methods for *regularization*, whereby one alters the network architecture or training process in order to decrease the generalization error of the model, without increasing its training error [23]. A common regularization method is data augmentation, where the

training data is slightly altered each time it appears in a mini-batch, in order to emulate the addition of more data [23]. Common augmentations in the case of image data include random cropping, flipping and rotation, as well as methods which alter photographic properties of the data, such saturation and hue. Early stopping is another method of regularization, whereby one stops the training of the network just as overfitting is beginning to take place (as measured by some stopping criterion), and choosing the model resulting from the best epoch as the training result.

2.2.2 ResNet

Another consideration when training deep neural networks is the problem of vanishing and exploding gradients [22]. As networks grow deeper, the numbers of computations during forward-propagation and backpropagation increase accordingly. If many of the gradients are either close to zero or have very large magnitude, then they will contribute to the gradients either tending towards zero (vanishing) or infinity (exploding) during back-propagation, which would effectively ruin the training process. With the intention to mitigate some of the problems with deeper neural networks, the concept of deep residual learning was introduced, along with the *ResNet* architecture [2]. ResNet is a type of convolutional neural network which contains *residual blocks*, as depicted in Figure 2.5.

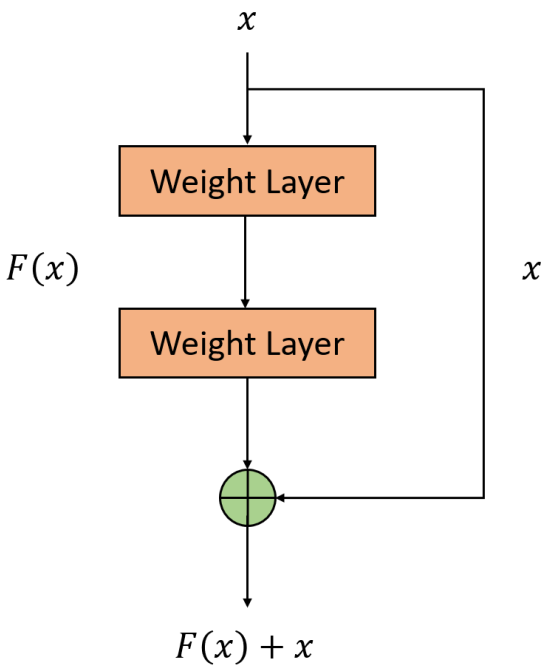


Figure 2.5: A residual block. An input **x** is passed through two weight layers with a ReLU activation in between and then added to the output of those layers.

In the above figure, a residual block is shown to consist of a set of weight layers, along with a *skip connection*. In addition to the standard connections between subsequent convolutional layers, a skip connection connects layers which are not in immediate

sequence, by adding the input to the residual block to its output. Implicit in the above figure are the ReLU-activations following the first weight layer and the skip connection. In other words, given an input $\mathbf{x}$ and a function R computed by the convolutional layers of the residual block, the output F of the residual block is

$$F(\mathbf{x}) = R(\mathbf{x}) + \mathbf{x} \iff R(\mathbf{x}) = F(\mathbf{x}) - \mathbf{x}.$$

As such, the learned function R is the difference between the input $\mathbf{x}$ and output $F(\mathbf{x})$, the so-called *residual*. The motivation for this is that the residual should be simpler for the network to learn, as compared to the entire transformation. The skip connections also create shortcuts during both the forward and backward pass, which, in theory, could reduce the effects of vanishing/exploding gradients, as it creates paths with a smaller number of multiplications. In practice, the addition of the residual blocks proved to be very successful at dealing with the downsides of increasing depth, and ResNet is now an architecture very commonly used as backbone in many different types of computer vision tasks.

For future reference, ResNet architectures of various depths are often referred to as e.g., ResNet-18, where 18 refers to the number of layers contained in the architecture. In this thesis, ResNet-18 and ResNet-34 are considered, and their layer structures are shown in Table 2.1.

Block	**ResNet-18**	**ResNet-34**	**Output Size**
Conv 1	64 kernels of size 7×7 and stride 2		$\frac{H}{2} \times \frac{W}{2}$
Pooling 1	Max-pooling of size 3×3 with stride 2		$\frac{H}{4} \times \frac{W}{4}$
Conv 2	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\frac{H}{4} \times \frac{W}{4}$
Conv 3	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\frac{H}{8} \times \frac{W}{8}$
Conv 4	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\frac{H}{16} \times \frac{W}{16}$
Conv 5	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\frac{H}{32} \times \frac{W}{32}$

Table 2.1: Table showing the ResNet-18 and ResNet-34 architecture, adapted from [2]. The leftmost column indicates the name of the block. Each element of an array denotes the kernel size followed by the number of kernels of a convolutional layer in the corresponding block. The rightmost column indicates the spatial dimensions of the feature maps produced by the block in each row.

In the above table, the rightmost column shows the spatial dimensions of the output

feature maps of each row, given initial spatial dimensions of $H \times W$. Note that after the first weight layer of each block, a downsampling by a factor of two is performed. As such, the final output feature maps of the ResNet are downsampled by a factor of 32. In [2], the convolutional layers are followed by a global average pooling layer and a fully connected layer with 1000 units.

2.2.3 Batch Normalization

Batch normalization [33] is another method to mitigate the effects of vanishing and exploding gradients. Batch normalization standardizes output features with respect to the batch mean and batch standard deviation of the feature. In addition to this, it incorporates learnable scale and variance parameters, which are initialized to 1 and 0 and are multiplied and added to the standardized outputs of the batch normalization, respectively. This means that batch normalization introduces a learnable standardization mechanism, which in theory will help the network learn faster by adapting output distributions, in addition to avoiding the vanishing/exploding gradient problem. However, the original motivation behind its effectiveness has been debated in more recent work. It has more recently been shown that batch normalization helps with training by smoothing the loss function [34]. Nevertheless, batch normalization has proven to be a very effective tool in the training of deep neural networks, and is very commonly seen in many state-of-the-art architectures.

2.2.4 Uncertainty Estimation

Given the output of a deep learning model, it is often desirable to also have a measure of how certain that output is. Especially so in the context of autonomous vehicles: when deployed in a critical situation, having a measure of how certain a network is of its prediction is crucial.

Two main types of uncertainty in this context are *aleatoric* uncertainty and *epistemic* uncertainty [35]. Aleatoric uncertainty is the uncertainty induced by variations in data, such as resolution or noise, and thus affects the amount of information a model actually can extract from data. Epistemic uncertainty on the other hand, denotes the uncertainty of the model parameters, and in essence captures the capability of the model at handling out-of-distribution input data.

One approach to estimating the aleatoric uncertainty is by constructing the network such that its output parametrizes a probability distribution instead of outputting a single value. For example, one could specify a Gaussian distribution and interpret the output of the network as the mean and standard deviation of the target distribution.

2.3 Attention

The concept of attention in the context of deep learning is the mechanism by which a network can selectively "attend" to different parts of an input or feature to varying

degrees. Originally appearing in the context of NLP [36], it has since seen applications in a number of machine learning fields. As an example of the usefulness of attention, consider the task of language translation. When translating a single word in a sentence, context and correlations between the current word and the rest of the sentence are highly relevant. Take Figure 2.6 as an example.

Figure 2.6: Toy example of attention. When processing the sentence in the illustration, attention captures that the words "car" and "driving" are strongly tied.

The attention mechanism quantitatively calculates a vector of weights denoting the importance of each other word in relation to the current word being translated, allowing the network take more global properties such as context into account. In the above figure, we see how the word "car" attends strongly to the word "driving", reflecting that these words bear meaning in relation to each other and therefore offer contextual clues.

The attention weights are calculated using a function which assigns a score to the other words in the sentence reflecting to what degree each word should be attended. In this work, a variation of the query/key/value-form of attention is used, inspired by the attention mechanism used in the Transformers architecture [37]. This particular form of attention can be described as follows: based on an input sequence $\mathbf{x}$, one generates via linear projection a set of query vectors $\mathbf{Q}$, key vectors $\mathbf{K}$, and value vectors $\mathbf{V}$, written in matrix form as

$$\begin{aligned}\mathbf{Q} &= \mathbf{W}_Q\mathbf{x},\\ \mathbf{K} &= \mathbf{W}_K\mathbf{x},\\ \mathbf{V} &= \mathbf{W}_V\mathbf{x},\end{aligned}$$

where the weight matrices $\mathbf{W}_Q$, $\mathbf{W}_K$ and $\mathbf{W}_V$ are comprised of learnable parameters. Each row in the above matrices correspond to one such vector, so that each matrix contain a number queries/keys/values. The attention weights are then calculated from the queries and keys as

$$\text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)$$

where $\sqrt{d_k}$ is the dimension of the key vectors, which is included in order to stabilize the Softmax function for high-dimensional keys. Note that the matrix multiplication inside the Softmax-function above implies that one computes the dot product between each query vector and every key vector. If these vectors *align*, then the dot product results in a large activation of the Softmax-function, and vice versa, which determines the magnitudes of the attention weights. The attention weights are then applied to the value vectors such that the total output of the attention mechanism is given by

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V},$$

and as such, the output is a weighted average of the input, with weights determined roughly by how well the corresponding query and key vectors align. In an analogy with a database, one could imagine functions, which given a search term could produce corresponding queries and keys. In this context, the query would take the shape of a delta function, such that when multiplied with the corresponding key, would return a vector which could be used to find the search result.

In the Transformer [37] architecture, the recurrent structure of RNNs and LSTM with an encoder-decoder structure which is based on an attention mechanism similar to the one above. Specifically, the Transformer employs *multi-head attention*, whereby the mechanism described above is applied several times in parallel, and the outputs are concatenated and linearly projected in order to produce the final output. In the context of NLP, this allows the model to capture multiple types of dependencies between different words in a sequence. Transformers-inspired language models such as BERT [38] and GPT-3 [30] have subsequently shown very impressive results and revolutionized the field of NLP, resulting in transformers becoming a mainstay of the NLP field. Their success in the field of NLP has also inspired their migration into other deep learning areas. Recently, the transformer architecture has been applied to vision [39] and object detection [40], with promising results.

2.3.1 Attention in Computer Vision

Since its introduction, attention has also seen applications in the field of computer vision. Due to its local nature, the convolutional operation has trouble capturing more global features of an image and correlations between pixels, which is the prime motivation for applying attention in this context.

In [41], the concept of self-attention is applied to convolutional neural networks. In this context, self-attention refers to the fact that the attention mechanism relates pixels to pixels in the same feature maps produced by the CNN. The mechanism is illustrated in Figure 2.7.

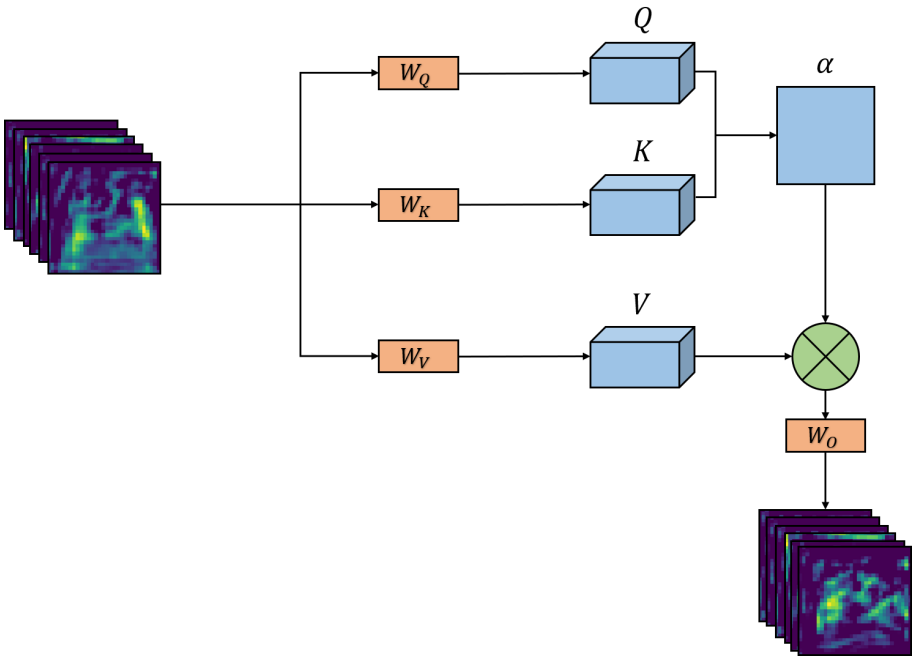


Figure 2.7: Illustration of the self-attention for vision used in [41]

In the figure, the triplet of tensors $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$ are denoted as the *query, key* and *value*, respectively. Given a set of feature maps $\mathbf{f}$ generated by a CNN, the triplet is calculated as

$$\begin{aligned} \text{Query} : \mathbf{Q} &= \mathbf{W}_Q \mathbf{f} \\ \text{Key} : \mathbf{K} &= \mathbf{W}_K \mathbf{f} \\ \text{Value} : \mathbf{V} &= \mathbf{W}_V \mathbf{f}, \end{aligned}$$

where the trainable weight matrices $\mathbf{W}_Q$, $\mathbf{W}_K$ and $\mathbf{W}_V$ are implemented as 1×1 convolution layers. Using the key and query, the corresponding attention map $\boldsymbol{\alpha}$ is then computed as

$$\boldsymbol{\alpha}_{(i,j)} = \text{Softmax}\left(\sum_c \mathbf{K}_{(i,c)} \mathbf{Q}_{(j,c)}\right)$$

Here, the c index corresponds to the feature channels, while the pixel coordinates (u, v) in $\mathbf{K}$ are flattened to one index, i, and $\mathbf{Q}$'s pixel coordinates are flattened to j. Furthermore, the Softmax operation is taken in the i-dimension.

The attention map is then applied to the value tensor, which is subsequently linearly projected to the appropriate dimensionality via another weight matrix $\mathbf{W}_o$. The attention maps reflect the relevance between pixels for the task at hand, and since these maps are computed globally, then this allows the network to learn more global feature correlations, which is otherwise difficult for the local convolution operation.

2.4 Perception In Autonomous Driving

In this section, we present relevant theory and methods for perception in autonomous driving.

2.4.1 Object Detection

Object detection is an area of computer vision dedicated to the detection of objects in visual data, such as images and video. This is a crucial feature in autonomous driving, as the detection and classification of surrounding objects is a key element in the planning and execution of safe driving. Given input data representing a scene, the goal of an object detection model is to draw *bounding boxes* around every object contained in the image, and also classify each bounding box according to which type of object they contain. Although there exist many different types of object detection methods, with the rise of deep learning, many of the best-performing object detection models have been deep learning-based.

More formally, given an input image containing a set of n objects $o_i \in \mathcal{O}$, the goal of a 2DOD model is to output a set of bounding boxes $b_i \in \mathcal{B}$ (where $b_i = (x_{\min}, y_{\min}, x_{\max}, y_{\max})_i$ in the two-dimensional case) such that the boxes encapsulate each object as tightly as possible in the image plane, along with a classification score c_i denoting the type of object detected.

2.4.2 3D Object Detection

In order to build 3D object detection models which can detect and locate objects in 3D, one needs to define how the model represents the shape, size, orientation and location of the object. In this work, we follow common practice in the literature by representing 3D objects as 3D boxes, with free parameters

$$\text{3D Box Parameters} = \begin{cases} \text{height, } h \\ \text{width, } w \\ \text{length, } l \\ \text{3D center, } \mathbf{p} = (x, y, z) \\ \text{yaw angle, } \alpha_{\text{yaw}} \\ \text{object class, } c. \end{cases}$$

In this work, we numerically represent quantities such as the above, measuring distances in units of meters, and angles in radians. Note specifically that, similarly to what is done on the KITTI benchmark [1], we do not account for the pitch and roll of vehicles in this work. Figure 3.11 illustrates a 3D bounding box, along with our use of notation.

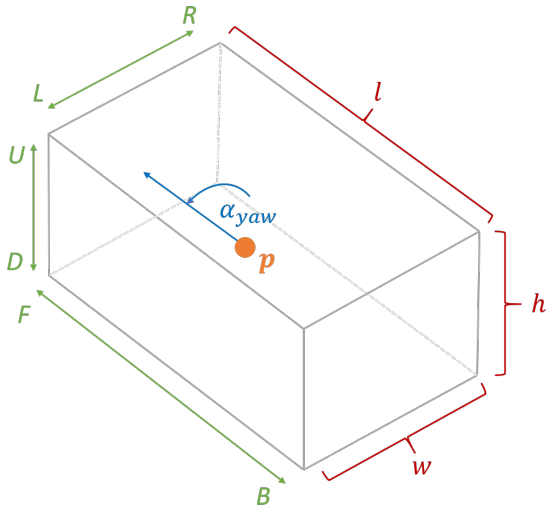


Figure 2.8: Figure illustrating a 3D bounding box. Here, **p** (orange) is the center position, α_{yaw} (blue) is the yaw angle, while the height h, width w and length l of the box are shown in red. We also illustrate where up, down, left, right, front and back lie along the box, by U, D, L, R, F, and B. Any combination of $\{U \text{ or } D,\ L \text{ or } R,\ F \text{ or } B\}$ thus denotes a corner of the box.

In the above figure, we also illustrate how the different corners of a bounding box are denoted. For example, we refer to the front upper right corner as FRU.

The overarching goal of building a 3D Object Detection model is to detect and locate objects in a scene. For each detected object, the model outputs class confidences and a 3D bounding box which should encapsulate the object. The end goal is that the model outputs these 3D bounding boxes such that their 3D volumes share a large overlap with the physical objects from the scene in 3D space, while simultaneously managing to correctly and confidently classify the type of object. Relevant literature and related work, with regards to 3D object detection will be presented in Section 2.7 on related work.

2.4.3 Evaluation Metrics

Given a set of bounding box predictions $\mathcal{B}$, it is desirable to measure how accurate these detections are with respect to the set of ground truth objects. To this end, one can categorize each detections as a true positive (TP), false positive (FP) or false negative (FN). True positive detections are defined as those detections which match a ground truth box according to some criteria (to be defined below). False positives can be defined as those predicted bounding boxes which do *not* match any ground truth bounding boxes in the data set, and false negatives as those ground truth bounding boxes for which no matching predictions have been produced. True negatives have no clear meaning in the object detection task, as each ground truth bounding box is assumed to contain an object.

Using this detection categorization, one defines the *recall* as

$$\texttt{Recall} = \frac{TP}{TP + FN}.$$

In words, the recall metric measures how many objects in a data set are actually found by the model. However, a model can achieve a high recall simply by outputting as many detections as possible, such that many of them happen to coincide with an object. To this end, *precision* is defined as

$$\texttt{Precision} = \frac{TP}{TP + FP}.$$

The precision metric measures the likelihood that a detection produced by the model actually corresponds to an object. In combination, precision and recall therefore provides a good measure of the overall performance of an object detection model.

In the classification setting, one can vary the model recall manually, by varying the confidence threshold above which a detection is considered a positive. Using this, one can vary the confidence threshold from 0 (where all model detections are positive) to 1 (where no model detections are positive), and at each level, note the resulting precision and recall values of the set of detections. One then acquires a set of precision/recall pairs. The procedure of varying the confidence threshold can be done by ranking all detections according to their associated confidence scores, from high to low. For each position in this ranked list, the recall and precision is computed for all objects ranked above the position.

Having computed such a set of precision/recall-pairs, one can compute the *average precision* metric by calculating the average precision value across all pairs:

$$AP = \frac{1}{n} \sum_{i=1}^{n} \rho(r_i),$$

where r denotes a recall value, and $\rho(r)$ is the corresponding precision value. An extension, the $AP|_{R_{40}}$ metric, which is an interpolated average precision metric commonly used for model evaluation on the KITTI [1] benchmarking data set, is defined as

$$AP\Big|_{R_{40}} = \frac{1}{40} \sum_{r \in R_{40}} \max_{r' \geq r} \rho\left(r'\right),$$

where R_{40} are 40 equidistant sampled recall positions, with exclusion of the zero-recall position $r = 0$. Furthermore,

$$\max_{r' \geq r} \rho\left(r'\right)$$

is the largest precision value observed above the recall threshold r.

As stated above, matching between detections and ground truths is done with respect to some criterion. For object detection, this criterion is most commonly the

intersection over union (IoU) metric. The IoU between two volumes V_1 and V_2 is, as implied by its name, defined as

$$\text{IoU} = \frac{V_1 \cup V_2}{V_1 \cap V_2}.$$

The IoU metric is equal to 1 when to bounding boxes overlap perfectly, and 0 when they are maximally distinct. By setting an IoU threshold, one can then categorize the set of detections as previously described. Note that the same definition holds if we exchange the volumes V_1 and V_2 for two areas, A_1 and A_2. As such, it is applicable to both 2DOD and 3DOD. One can also define the *birds eye view* (BEV) IoU as the intersection between the detected- and ground truth box, viewed as rectangles from a top-down projection in the camera frame.

In this work, we evaluate model performance using two metrics based on the above theory. Firstly, we consider the $AP|_{R_{40}}$ metric in the BEV setting, with the top-down projection IoU threshold set to 0.5. We also consider the $AP|_{R_{40}}$ metric using the relative center error (RCE), defined as

$$\text{RCE} = \frac{\|\mathbf{p} - \mathbf{p}_{\text{GT}}\|}{\|\mathbf{p}_{\text{GT}}\|},$$

where $\mathbf{p}$ and $\mathbf{p}_{\text{GT}}$ are the vectors to the center of the detection and ground truth-box, respectively. Using this measure, true positives are detections with an RCE below 5%. This allows for a more relaxed measure of detection quality, which is suitable for longer distances where a high IoU might be very hard to achieve, but where there might be relevance in measuring how far off a detection is.

2.5 Monocular Camera Data

The raw data in our data set consists of RGB camera data. Due to confidentiality agreements, it will have to suffice to mention that the camera being used can be approximated using the spherical Kannala camera model [42]. In this work, we represent spatial data in the *camera frame*, or camera coordinate system, illustrated in Figure 2.9.

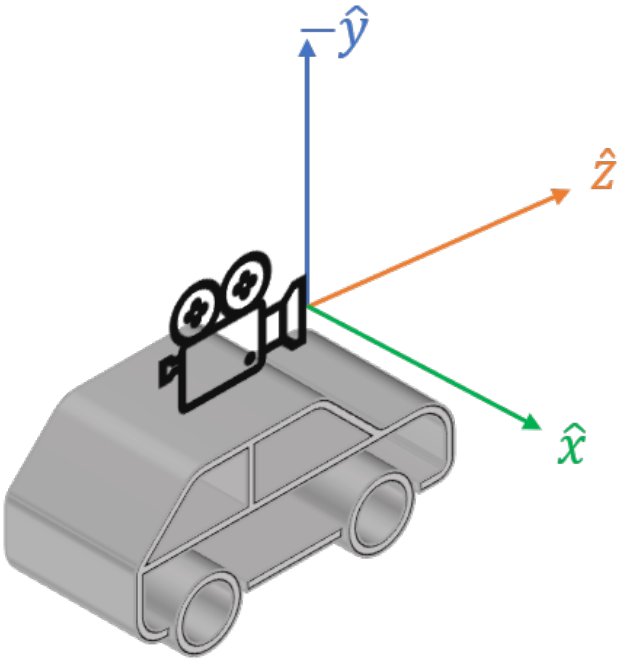


Figure 2.9: Illustration of the camera coordinate system, or camera frame. A camera is placed on the top of the car, pointing forward. Note specifically, that the y-axis points downward, such that the coordinate system is right handed.

The rest of this section will cover how we calibrate our camera data, as well as how we transform between different coordinate frames using these calibrations, in conjunction with the aforementioned Kannala camera model.

2.5.1 Calibration

It is important to highlight that the camera frame may differ between different photography sessions, as a result of the mounting position of the camera being adjusted between sessions. However, such differences can accounted for by means of an extrinsic calibration of the camera position,

$$\texttt{Extrinsics} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Here, R_{ij} represent the components of a rotation matrix, while t_x, t_y, t_z represent the components of a translation vector. If we want to transform a vector $[x, y, z]^T$ using the extrinsic calibration matrix, we can do this by adding a one as the fourth component of the vector, $[x, y, z, 1]^T$. Applying the extrinsic calibration matrix from the left will then simultaneously rotate and translate the vector to the calibrated coordinates $(x', y', z', 1)$, after which we can read off (x', y', z'). In our data set there is an extrinsic calibration matrix for each data point, specifying the relation between

the camera frame and an invariant coordinate system in the car. Therefore, we can restrict ourselves to always working in the camera frame coordinates.

In addition to the extrinsic calibration, we also have access to an intrinsic camera calibration for each image, given by the parameters

$$\texttt{Intrisics} = \begin{cases} f_x, f_y, \\ [c_u, c_v], \\ l_1, l_2, l_3, l_4. \end{cases}$$

Here, f_x and f_y are the focal lengths in the x and y axes in the pixel coordinates, $[c_x, c_y]$ is the optical center, and $l_1, ..., l_4$ are the undistortion coefficients.

2.5.2 Unprojecting 2D to 3D

In this thesis, our data set is collected using a spherical camera model. Using the Kannala model [42] for approximating the spherical camera, we can unproject a point (u, v) in the pixel coordinates to a point (x, y, z) on the unit sphere $x^2 + y^2 + z^2 = 1$. From the Kannala model we have

$$\begin{aligned} u' &= (u - c_x)/f_x \\ v' &= (u - c_y)/f_y, \\ \rho &= \sqrt{u'^2 + v'^2}, \\ \phi &= \arctan\frac{v'}{u'}, \\ \theta &= \rho\left(1 + l_1\rho^2 + l_2\rho^4 + l_3\rho^6 + l_4\rho^8\right). \end{aligned}$$

Using these relations, the Cartesian coordinates on the unit sphere can be expressed as

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix}. \tag{2.3}$$

If we have assigned a depth value z' in the camera frame to the pixel coordinate (u, v), we can use Equation 2.3 to calculate the norm

$$\sqrt{x'^2 + y'^2 + z'^2} = \frac{z'}{\cos\theta}.$$

As such, we get a closed expression for unprojecting a depth value z' in the pixel coordinates (u, v) to the physical camera frame from Figure 2.9,

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = z' \begin{pmatrix} \tan\theta\cos\phi \\ \tan\theta\sin\phi \\ 1 \end{pmatrix}. \tag{2.4}$$

2.6 Data Representations

In this work, we make an important distinction between different data representations. Specifically, we distinguish between representations used for image- and point cloud data. The importance of this distinction relates to the type of neural networks which can be used to process the different representations. For point cloud representations, special architectures have to be employed to account for the unordered nature of the data, whereas image data representations enable the use of powerful convolutional neural networks.

2.6.1 Point Cloud Representations

LiDAR sensors produce data containing spatial information about the environment surrounding the sensor. By using time-of-flight, measured by sending out light beams and measuring the time until a reflected beam returns to the sensor, one can calculate a point in 3D space which lies on the reflective surface. This process result in a set of points called a *point cloud*. Formally, point clouds are defined as unordered sets of points with continuous values

$$\{(x_1, y_1, z_2), ..., (x_N, y_N, z_N)\} \tag{2.5}$$

which describe the surfaces 3D environments. In part, LiDAR sensors are useful due to their high precision [18].

In [18], the performance differences between LiDAR-based and mono-depth based 3DOD methods is analyzed. It is concluded that the unsatisfactory performance of mono-depth based approaches is not only due to the quality of the depth estimations, but also a question of data representation. By incorporating the predicted depth values from a mono-depth estimator into a so-called *Pseudo-LiDAR* representation and applying standard LiDAR-based methods, the performance gap is significantly reduced. The representation is achieved by unprojecting the pixel-wise depth to a Pseudo-LiDAR point-cloud, represented analogously to Equation 2.5. More formally, a depth estimator is used to produce a depth map, and the unprojection is conducted with respect to the formalities described in Section 2.5.2, where the the intrinsic and extrinsic parameters of the camera are used to move from pixel-pixel-depth space, to physical Cartesian space.

2.6.2 Image Data Representations

In [19], it is shown that the Pseudo-LiDAR representation is not an essential factor for mono-depth based 3DOD, and that the *image representation* is also feasible. Contrary to the unordered structure of a point cloud representation, the image representation of a point cloud consists of points ordered by their pixel position. As such, a Pseudo-LiDAR point cloud may be converted to the corresponding image representation, by considering each spatial coordinate as a separate input channel in the image representation: given a set of pixel-wise depth estimations for an image, one unprojects each pixel to acquire pixel-wise x and y-coordinate estimations. By

keeping these estimations ordered according to their corresponding pixel-locations, one obtains a set of three input maps, much like the red, green and blue channels of an RGB image. In [19], it is further shown that this enables the use of more powerful CNN backbone networks, as is the standard for image data.

2.7 Related Work

This section aims to give a brief introduction to depth estimators, while also providing an overview of existing methods for object detection. We cover representative approaches based on RGB images, LiDAR point clouds as well as novel Pseudo-LiDAR based approaches.

Throughout this section, and the rest of the report, illustrations make use of colors to aid the reader in understanding the functions of different modules. To clarify, the colors of modules relate to their function as shown in Figure 2.10.

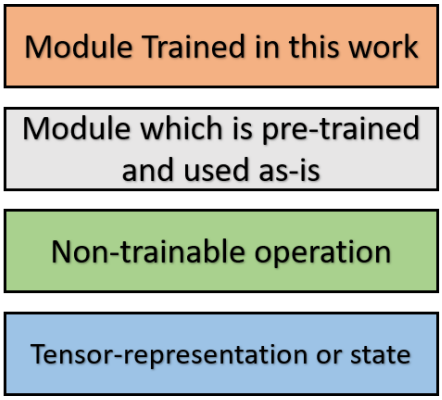


Figure 2.10: Module color coding.

2.7.1 Depth Estimation

The task of predicting the dense (pixel-wise) depth map from a single RGB-image is commonly referred to as *monocular depth estimation*, also known as mono-depth estimation. It is an inherently ill-posed problem, as any given 2D image may correspond to many different 3D scenes [12]. Although progress in mono-depth estimation has been slow compared to e.g., stereo-based methods, deep learning based models have recently shown promising results mono-depth estimation [43, 12, 13]. Semi-supervised approaches to mono-depth estimation have also seen rapid development in recent years [44].

The *Deep Ordinal Regression Network* [12] (DORN) architecture is one of the mono-depth estimators which has seen applications in the monocular 3D object detection literature [18, 19]. The network architecture is illustrated in Figure 2.11.

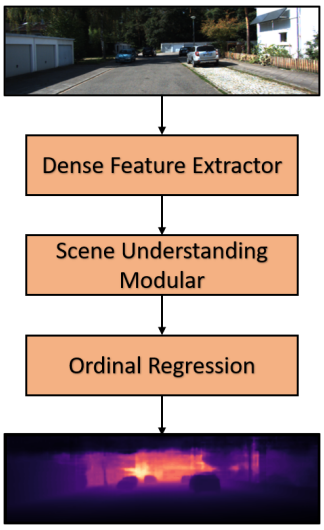


Figure 2.11: High-level illustration of the DORN Architecture. A monocular RGB image is sequentially processed by a dense feature extractor and scene understanding modular, after which *ordinal regression* is performed.

In the above figure, we see that the input monocular RGB image is sent to the dense feature extractor which utilizes *dilated convolutions*, an alternative operation to traditional convolutions which enlarge the field-of-view of the filters, without decreasing spatial resolution or increasing the number of parameters. The scene understanding modular uses both pooling layers and fully connected layers, to obtain multi-channel pixel-wise features. Finally, *ordinal regression* is used to produce the output pixel-wise depth-map.

Ordinal regression approximates the continuous depth axis by using a set of discrete, log-spaced bins. When fed an image, the model outputs the cumulative distribution over the bins, that a given pixel should lie in that bin or closer. During inference the output depth is chosen as the center of the bin for which the cumulative distribution exceeds 50%.

2.7.2 Image-based 2D Object Detection

Many of the models which will be discussed in the coming sections utilize 2DOD networks, since these networks are often highly accurate at detecting objects in 2D images. For example, Frustum PointNet[12] utilizes a 2D detector to propose regions containing objects, and uses this information for accurate LiDAR based 3D object detection. You Only Look Once [45] (YOLO) is a representative CNN based 2DOD architecture. The architecture is illustrated in Figure 2.12 at a high level of abstraction.

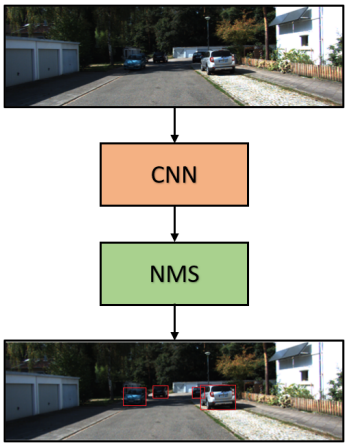


Figure 2.12: High-level illustration of the YOLO Architecture. An image is processed by a CNN, and *non-maximum suppression* (NMS) is applied, producing a set of detections.

At the input, a grid is applied to the image, dividing it into a set of cells. Each cell is assigned those ground truth objects which have their centers within that cell, and is tasked with detecting those objects. As such, each grid cell produces B bounding boxes and confidence scores, intended to capture the objects belonging to that grid cell. After feeding the input image through the CNN-based network producing the box proposals, *non-maximal suppression* (NMS) is applied. NMS is a method of selecting a subset of detections to be kept if there are many overlapping detections. In contrast to other CNN-based detectors such as R-CNN, YOLO introduces additional spatial constraint on the grid cell proposals, yielding fast inference speeds.

2.7.3 LiDAR + Image-based 3D Object Detection

3D object detection pipelines utilizing LiDAR point clouds currently outperform other approaches on the popular KITTI benchmark data set [1]. The Frustum PointNets (F-PointNets) architecture [7], illustrated in Figure 2.13, is one of the representative architectures utilizing the high precision of LiDAR.

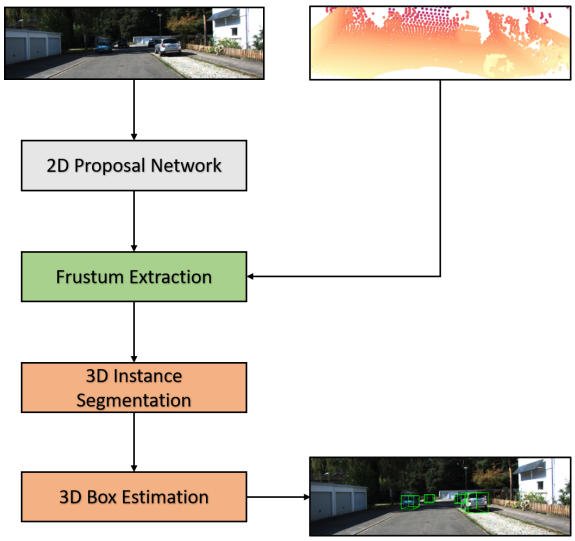


Figure 2.13: High-level illustration of the Frustum PointNet [7] Architecture. Images in the illustration are taken from the KITTI train set [1]

In [7], both monocular RGB images and LiDAR are utilized. The excellent performance of state-of-the-art CNN-based 2DOD model is leveraged to propose a set of 2D regions along with content classifications (2D Proposal Network in figure). These regions are then unprojected to 3D space as cone-like slices called *frustums*, and used to crop out a subset of the LiDAR point cloud in the frustum extraction step. A *PointNet* [46], a point-wise backbone network designed to handle 3D point clouds and accounts for varying number of points, is used in the 3D instance segmentation step, aimed at a second stage filtration of points not belonging to the object. Finally, another PointNet is fed the remainder of the point cloud which was assigned to the object, outputting the final 3D bounding box parameters. Note that for this architecture, the 2DOD network is pre-trained and treated as is. As such, frustums of points are generated in advance using the detections produced by the 2DOD network, and only the 3D-detection related parts of the pipeline are trained.

Aggregate View Object Detection (AVOD) [8] is another representative 3D object detection architecture, shown in Figure 2.14, which also utilizes both RGB image and LiDAR point clouds.

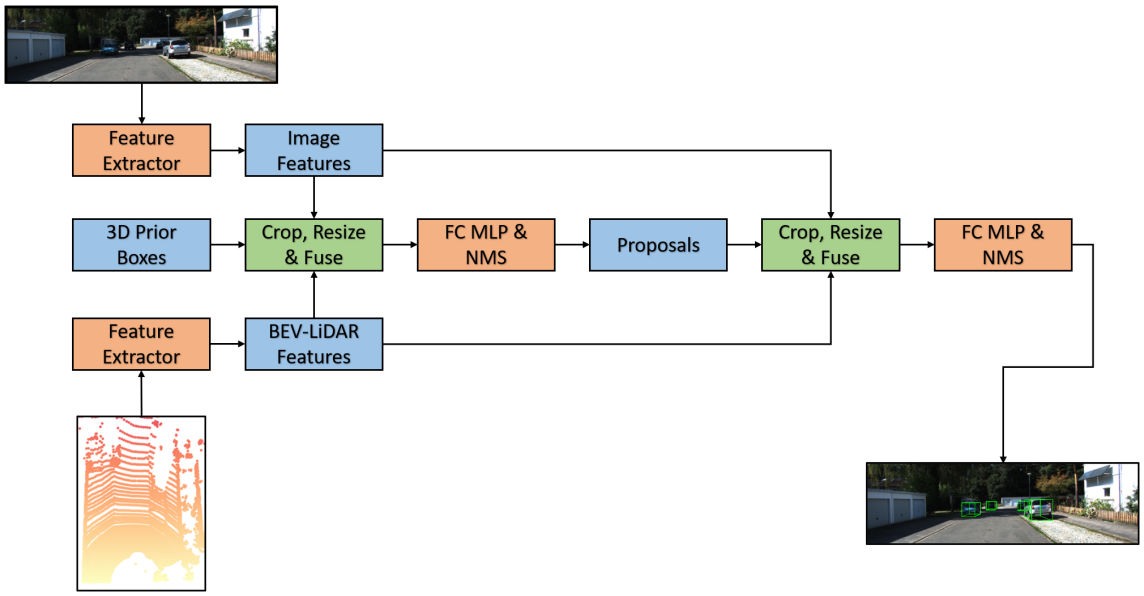


Figure 2.14: High-level illustration of the AVOD [8] Architecture. Images in the illustration are taken from the KITTI train set [1].

The AVOD architecture fuses high resolution features extracted from monocular RGB images with features extracted from LiDAR point clouds viewed from the birds-eye-view perspective. For each box from a prior 3D anchor grid, the projected regions in the image- and BEV-LiDAR-features are cropped, resized and fused. The fused features are passed through a fully connected MLP, and NMS is used to keep a set of top proposals. After this, a second stage refinement of the proposed boxes is conducted by repeating the cropping and resizing process. Using another MLP followed by NMS, a final set of 3D bounding boxes are proposed. For this architecture, in contrast with Frustum PointNets architecture described above, all parts of the pipeline are trained together.

2.7.4 Image-based 3D Object Detection

In [17] the *Multi-Level Fusion* (MLF) architecture for monocular image based 3DOD is proposed, shown in Figure 2.15.

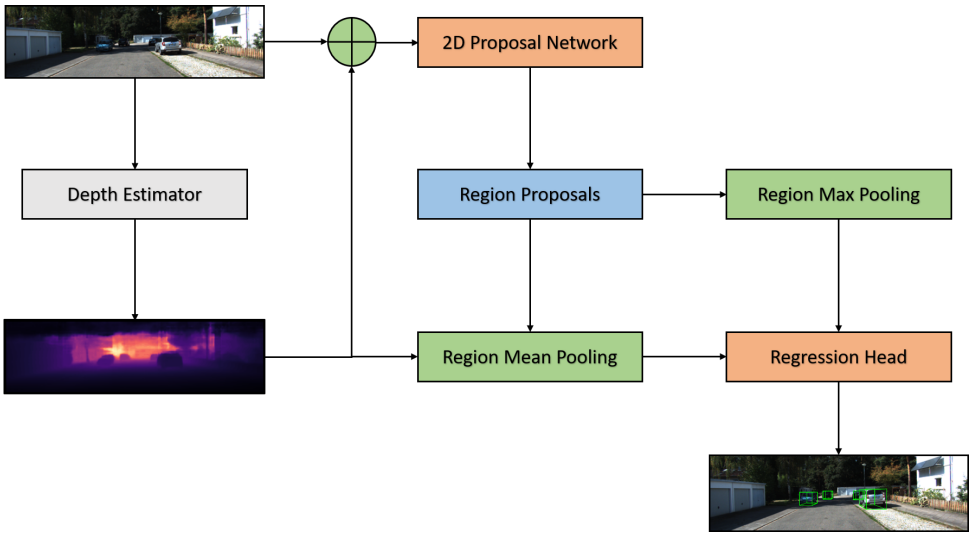


Figure 2.15: High-level illustration of the Multi-Level Fusion Architecture [17]. Images in the illustration are taken from the KITTI train set [1]. Depth estimations were generated using the DORN [12] network.

In the MLF pipeline, a pre-trained network is used for estimating pixel-wise depth in the image. The image-plane depth information is then concatenated with the RGB channels from the image image, and used as input to a 2D proposal network. Using the proposed regions, the CNN features are max-pooled inside each region, producing a feature vector. Simultaneously, another feature vector is produced by average pooling the image-plane depth information inside the same region. Finally, the pooled features are fused and fed to an MLP which regresses the 3D bounding box parameters. Such MLPs shall henceforth be called *regression heads*. Note that for this architecture, the depth estimator is pre-trained and treated as-is, i.e., its parameters are not updated during training of the MLF pipeline.

In [18], it is hypothesized that a significant factor in the performance discrepancy between image-based depth-estimation- and LiDAR based architectures may be attributed to the choice of how the spatial data is represented. It is pointed out that LiDAR data is commonly represented either as a 3D point cloud or from a top-down view (BEV), the authors propose utilizing a depth estimation network to produce a pixel-wise depth map, and then unprojecting it using camera parameters to produce a Pseudo-LiDAR point cloud. This point cloud is then used as input to a LiDAR based pipeline, in conjunction with the original RGB image. The proposed pipeline is illustrated in Figure 2.16.

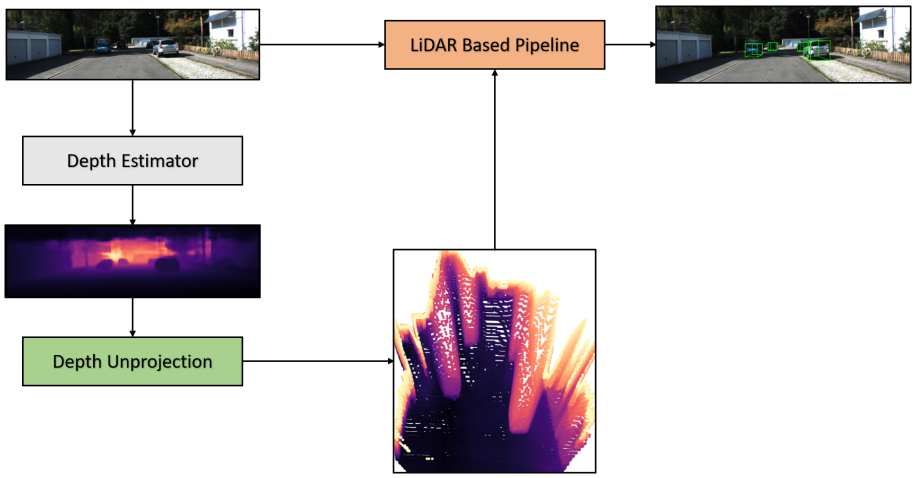


Figure 2.16: High-level illustration of the Pseudo-LiDAR architecture proposed in [18]. The large block in the bottom illustrates the Pseudo-LiDAR point cloud. Images in the illustration are taken from the KITTI train set [1]. Depth estimations were generated using the DORN [12] network.

In the paper the authors conduct experiments by using the AVOD and Frustum PointNets architectures, which were described in Section 2.7.3, while substituting LiDAR with the proposed Pseudo-LiDAR. Large performance gains were reported over previous image-based 3DOD pipelines, with some metrics increasing more than two-fold over previous state-of-the-art. The models are trained as was described in Section 2.7.3, while the pre-trained depth-estimator is treated as-is.

In [19], the root cause of the performance benefits of the Pseudo-LiDAR representation is more thoroughly investigated. It is shown that it is actually the coordinate transform used to unproject depth estimations from pixel coordinates (u, v, z) to 3D coordinate system (x, y, z) which is the key to the increases in performance. Since this transformation implicitly encodes the camera parameters into the data, it is argued that this is more important than the point cloud representation, allowing depth estimations to be converted to the image representation. Since one is no longer constrained to the point-wise networks otherwise thought essential for Pseudo-LiDAR, an architecture called PatchNet is proposed, showing that more sophisticated CNN backbones may be used instead. Figure 2.17 shows a high-level view of the proposed architecture, which has many conceptual similarities with the F-PointNets architecture.

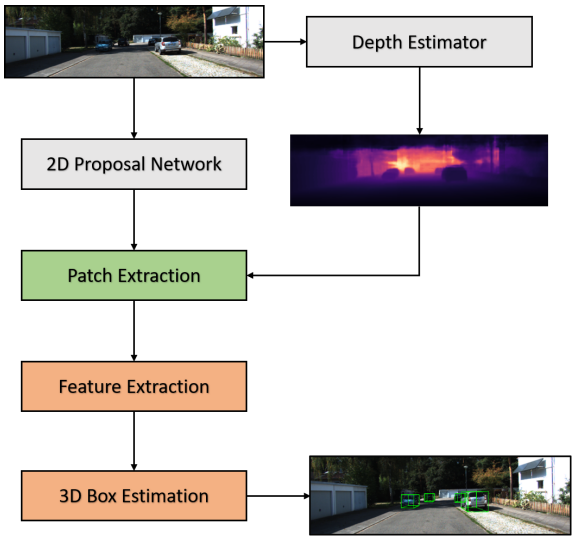


Figure 2.17: High-level illustration of the PatchNet Architecture. Note the similatities with the F-PointNets architecture. Images in the illustration are taken from the KITTI train set [1]. Depth estimations were generated using the DORN [12] network.

In the above figure, a pre-trained depth estimator estimates the pixel-wise depth in the input RGB image. The intrinsic and extrinsic camera calibration is then used to unproject the pixel-wise depth to 3D points while, in contrast to [18], maintaining the data in image representation rather than using the point cloud representation. As a result, one obtains *depth maps*, characterized by pixel-wise x, y and z values in the image representation. A CNN-based 2DOD network simultaneously produces a set of detections, forming 2D region proposals. For each detection, the corresponding region is cropped out from the depth maps, producing a *patch*. Each patch is then independently processed by means of a CNN-based feature extractor, after which the resulting feature vector is fed to a regression head which estimates the parameters of a 3D bounding box. Repeated for all patches, this produces the final set of 3D bounding boxes. During training, the 2DOD network and depth estimators are kept as-is, such that only the feature extractor and regression head are trained.

In [20], it is stated that there is a geographical overlap between the training examples used to train the depth estimators employed in [19] and [18] and the validation set used for the KITTI 3D object detection benchmark. As a result, performances reported using this set-up are found to be inflated. In addition to delving into this inflation in performance, the authors propose an architecture illustrated in Figure 2.18, building upon PatchNet [19].

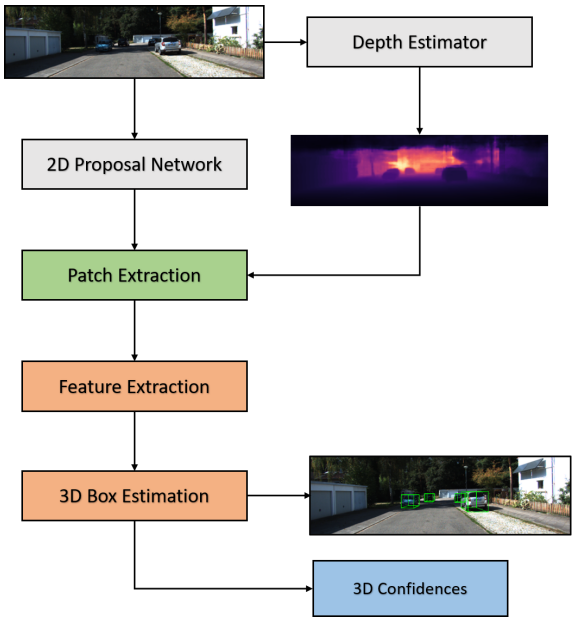


Figure 2.18: High level illustration of architecture proposed in [20]. Images in the illustration are taken from the KITTI train set [1]. Depth estimations were generated using the DORN [12] network.

The above figure shows how an additional 3D-confidence branch is added to the PatchNet architecture. It is shown that 3D object detection pipelines utilizing 2DOD bounding boxes as region proposals, such as PatchNet and F-PointNets, typically assign prediction confidences to the 3D predictions using the confidences generated by 2DOD network. Since the confidences play a crucial role in the $AP|_{40}$ metric used to evaluate and compare models on the KITTI object detection benchmark, a mechanism for outputting a dedicated 3D confidence prediction is proposed, which aim to better reflect the quality of the actual 3D bounding box.

2.7.5 Points Selection Process

In 3DOD architectures utilizing 2DOD region proposals, it is common to utilize a second-stage points selection process, to filter out data not belonging to the object of interest. In this section, the points selection processes of F-PointNets [7], and PatchNet [19] are described.

In the F-PointNets [7] architecture, this problem is solved by means of a separate point-wise *instance segmentation* network. In other words, given a frustum containing a LiDAR-point cloud, this point cloud is processed by a instance segmentation network which classifies each point in the point cloud as belonging to the object of interest or not. These points are then selected for further processing, while the points not classified as belonging to the object of interest are discarded. Given the high accuracy of LiDAR point clouds, this can result in highly accurate point selections. However, this type of mechanism also requires training of a separate instance segmentation, and consequently, point clouds which are labeled for instance segmentation.

In the PatchNet [19] architecture, the point selection process is handled by means of a binary mask (see Figure 1.3). Given a patch of depth estimations, the mean depth value of the patch is used as a threshold. Pixels with depth estimations above this threshold are assumed to belong to the background, while pixels with depth estimations below this threshold are considered foreground pixels and more likely to be relevant. From this, a binary mask is generated such that pixels with depth values below the threshold are set to 1, and pixels with depth values above the threshold are set to 0. This binary mask is then applied to the feature maps produced by the backbone network, effectively zeroing out the contribution of pixels which are discarded by the binary mask. The process is likened to a sort of hard attention mechanism. While this approach does not require the training of any additional networks, it does not handle the problem of occlusion optimally, often resulting in examples as shown in Figure 1.4. The figure shows how the binary mask disregards many pixels which actually belong to the object of interest, and also retains pixels belonging to the occluding car as well as background points.

2.7.6 Performance Comparison

Despite the advantages of image based 3DOD, there remains a large gap between state-of-the-art image based 3DOD models and LiDAR based approaches to date. However, the recent Pseudo-LiDAR approaches have yielded substantial improvements over previous image based approaches. Table 2.2 shows the performances of the representative models discussed in sections 2.7.3 and 2.7.4, as evaluated on the KITTI benchmark test set, where online evaluation is mandatory and the ground-truth objects are kept secret.

		Testing $AP\|_{40}$			Validation $AP\|_{40}$			Validation $AP\|_{11}$		
KITTI Car @ IoU 0.7		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Mono	MLF [19, 18]	7.08	5.18	4.68	-	-	-	10.5	5.7	5.4
	Pseudo-LiDAR [20]	14.17	8.47	7.29	24.47	13.40	10.92	28.2	18.5	16.4
	PatchNet [20, 19]	15.70	10.15	8.79	31.60	18.22	15.10	35.1	22.0	19.6
	PatchNet + 3D Conf. [20]	23.66	13.25	11.23	38.30	24.11	19.23	-	-	-
Stereo	MLF [19, 18]	-	-	-	-	-	-	-	9.8	-
	Pseudo-LiDAR [19, 18] -	-	-	-	-	-	-	59.4	39.8	33.5
	PatchNet [19]	66.0	41.1	34.6	-	-	-	65.9	42.5	38.5
LiDAR	F-PointNets [8]	<u>81.20</u>	<u>70.39</u>	<u>62.19</u>	-	-	-	<u>82.6</u>	<u>68.8</u>	<u>62.0</u>
	AVOD [8]	**81.94**	**71.88**	**66.38**	-	-	-	**82.8**	**73.5**	**67.1**

Table 2.2: Here, we summarize the performance of models described in sections 2.7.3 and 2.7.4 on the KITTI 3D object detection benchmark. We show performance in the easy, moderate and hard difficulty levels for the car category. *Mono* and *Stereo* denote whether the results were obtained using monocular and stereo camera data, respectively, while LiDAR implies that the model uses LiDAR and image data. The best results are highlighted in **bold**, and the runner-ups are <u>underlined</u>. A dash implies that no results were reported in the paper for the corresponding category. Citations indicate sources of numerical results in the respective row.

In the above table, we also included the $AP|_{11}$ metric on the evaluation set since it was conventionally used until recently, when it was highlighted in e.g., [47] that 11 recall positions inflated model performance and proposed to use finer interpolation. One needs to carefully digest this table, since as it was pointed out in [20], there is an overlap between the depth training set and the object detection validation set, affecting all models in the *Mono* and *Stereo* categories. Still, it is clearly evident from the above table that the current state-of-the-art 3DOD models are LiDAR based, almost doubling the performance of stereo image based models in some categories using the $AP|_{40}$ metric. The table also highlights how difficult the task of monocular image based 3DOD is due to the comparatively low performances, a difficulty often attributed to the lack of precise spatial information [18, 19]. In contrast to monocular imagery, stereo images provide more precise spatial information, a fact well understood by comparing the quantitative results in the respective categories.

3

Architecture

In this chapter, we describe the specific network architectures employed for our experiments. Firstly, sections 3.1, 3.2, 3.3 and 3.4.1 describe the PatchNet architecture. Modifications to this architecture will be clearly highlighted throughout these sections. Secondly, we describe how we replace the binary mask in PatchNet with an attention mechanism. This specific modification is described in detail in Section 3.5.

3.1 Overview

In this thesis we study how PatchNet performs in different settings, and how different modifications affect the performance of the model. Figure 3.1 illustrates the structure of PatchNet at a high level of abstraction, where our modifications are not visible. The pipeline receives a monocular RGB image at the input layer, and outputs a set of 3D bounding boxes, parametrized by their center position, height, width and length, as well as yaw angle.

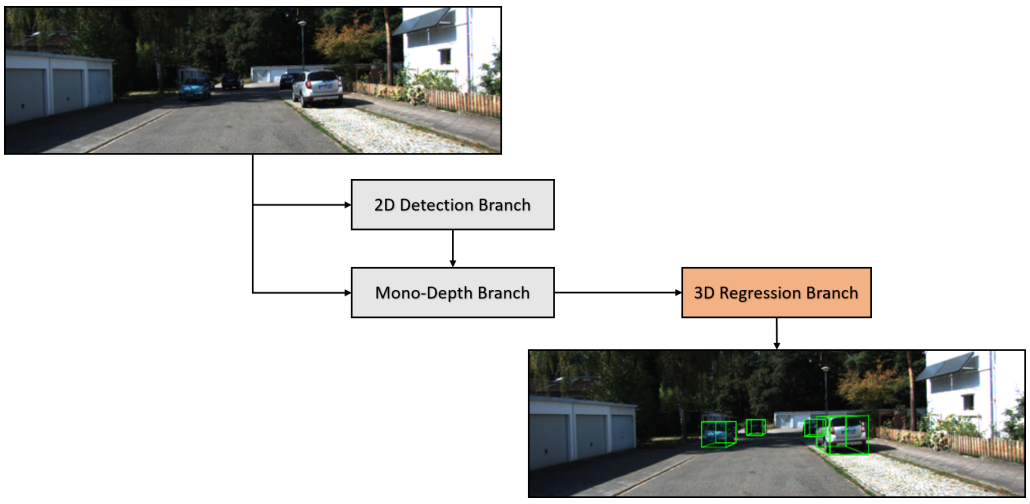


Figure 3.1: Overview of the PatchNet architecture. The proposed attention mechanism is not visible in this illustration. The illustration is at a higher level than Figure 2.17. Images in the illustration are taken from the KITTI train set [1].

The above figure shows the intermediate steps of the architecture. Given a full

monocular RGB image, the 2DOD and mono-depth networks are applied to this image. This results in pixel-wise depth estimation and a set of 2D bounding boxes. The areas of the input image and the depth estimations corresponding to the bounding boxes are subsequently cropped out to form *patches*. As such, each patch corresponds to one object. Each patch is then used as input to the 3D Regression Branch, which outputs the final 3D bounding boxes. In the end, one obtains a full set of 3D bounding boxes for all detected objects in the image.

Recall the color-coding used in our model diagrams (see Figure 2.10), and note specifically in Figure 3.1 that we only train the 3D Regression Branch. Since we are provided with pre-trained mono-depth estimation and 2DOD networks, all processing before the 3D Regression Branch can be done in advance to training the actual model. These steps will be described in more detail in the subsequent sections. For clarity, we emphasize once again that the primary novel modifications we make to the PatchNet architecture are also contained in the 3D Regression Branch.

3.2 2D Detection Branch

PatchNet takes inspiration from the F-PointNets architecture (Section 2.7), utilizing the high performance of modern 2D detectors to propose regions of interest in an input image. More concretely, in the 2D Detection Branch, the RGB image gets passed to a 2DOD network, which in turn produces a set of n detections $d_i \in \mathcal{D}$, as illustrated in Figure 3.2.

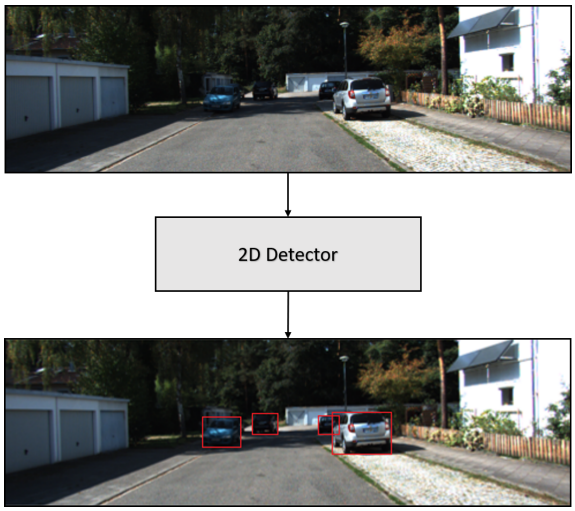


Figure 3.2: Abstraction of the 2D Detection Branch. A single stage 2DOD network produces a set of 2D bounding boxes with corresponding class probabilities. Here, detections are illustrated as red bounding boxes.

Each detection such as the ones illustrated above is characterized by a 2D bounding box and class probabilities.

For our baseline 2DOD network we employ a one-shot network based on a fully-

convolutional ResNet, which successively downsamples the feature maps, while increasing the number of feature channels, which are then fed to a feature pyramid network and finally to a set of classification and regression heads. We remind the reader that in addition to outputting 2D bounding boxes, the network also outputs 3D bounding boxes, which we use to make a direct comparison between the baseline network and our 3D Detection Branch.

As was also stated earlier, the 2DOD network is treated as is (similar to what is done in [7, 19, 20]). In other words, it is trained separately, and remains fixed throughout the study.

3.3 Mono-Depth Branch

The Mono-Depth Branch in our model is used to extract spatial information from the monocular input image, and crop out *regions of interest* (ROIs) corresponding to 2D bounding boxes from the 2D Detection Branch. This process is illustrated at a high level of abstraction in Figure 3.3.

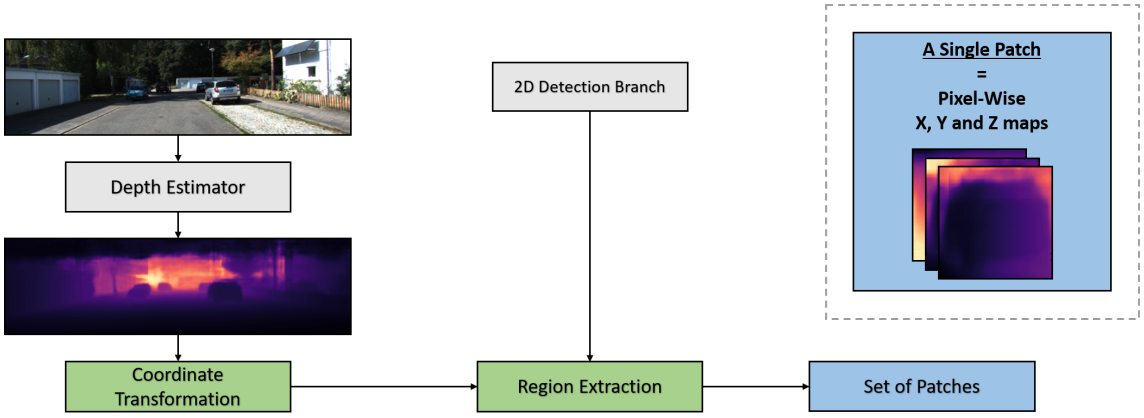


Figure 3.3: Abstraction of the Mono-Depth Branch. Images in the illustration are taken from the KITTI train set [1]. Depth estimations were generated using the DORN [12] network.

In the figure, we see that pixel-wise depth estimation is performed on the input image by means of a depth estimator. A coordinate transformation is then applied to the resulting depth map, after which ROIs are cropped out using 2D bounding box proposals from the 2D Detection Branch. The set of extracted regions, or *patches*, are the outputs of this branch. In the figure, we also emphasize that a single patch corresponds to the cropped pixel-wise x, y and z maps, and is therefore illustrated as three overlaid maps.

In the coordinate transformation step, the pixel-wise depth map is unprojected to 3D coordinates using Equation 2.4, yielding an image representation of the 3D

coordinates in the camera frame,

$$z(u,v) \longrightarrow \mathbf{r}(u,v) = \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{(u,v)} .$$

The region extraction step takes the map of pixel-wise coordinates, and a set of detections $d_i \in \mathcal{D}$ from the 2D Detection Branch. For each detection d_i, we produce exactly one patch, $\mathbf{p}_i$, consisting of the pixel-wise coordinates in the bounding-box region of d_i

$$\mathbf{p}_i(u,v) = \mathbf{r}(u,v), \quad u,v \in \mathbb{Z} : u \in \left[u_i^{\min}, u_i^{\max}\right], v \in \left[v_i^{\min}, v_i^{\max}\right],$$

where $u_i^{\min}$ and $u_i^{\max}$ denote u-axis limits of d_i, and similar for v. Repeating this for each detection in $\mathcal{D}$ produces the output of the Mono-Depth Branch, corresponding to the set of patches for the given image. The patch extraction step allows for more fine-grained subsequent treatment of the respective objects in the scene.

Note that all modules in the Mono-Depth Branch are either pre-trained or non-trainable. Due to confidentiality agreements, we can only disclose that the depth estimator is a comparatively lightweight deep neural network consisting of a simple CNN encoder-decoder with skip-connections and a regression head. In addition to pixel-wise depth estimates, the networks also outputs corresponding pixel-wise aleatoric uncertainty values. This uncertainty corresponds to the standard deviation of a learned probability distribution over the output depths. Here, we denote the pixel-wise depth as $z(u,v)$, due to its analogy with the z value in the camera frame.

3.4 3D Regression Branch

The 3D Regression Branch receives a single patch $\mathbf{p}$ as input, and outputs a 3D bounding box for this patch. Note that this implies that if an input image contains multiple objects which have been detected in the 2D Detection Branch, the 3D Regression Branch processes multiple patches to produce the entire set of 3D detections for the image.

Since our modifications to the PatchNet architecture are contained in this branch, we split this section in two parts. Firstly, Section 3.4.1 describes the 3D Regression Branch of PatchNet in more detail, clearly highlighting any changes made to the original architecture throughout the text. Secondly, Section 3.5 describes precisely how we replace the binary mask in PatchNet by means of an attention mechanism similar to the one described in Section 2.3.1.

3.4.1 PatchNet

Figure 3.4 illustrates the 3D Regression Branch of PatchNet pipeline, detailing how a single patch is processed in order to produce a single, corresponding, bounding box.

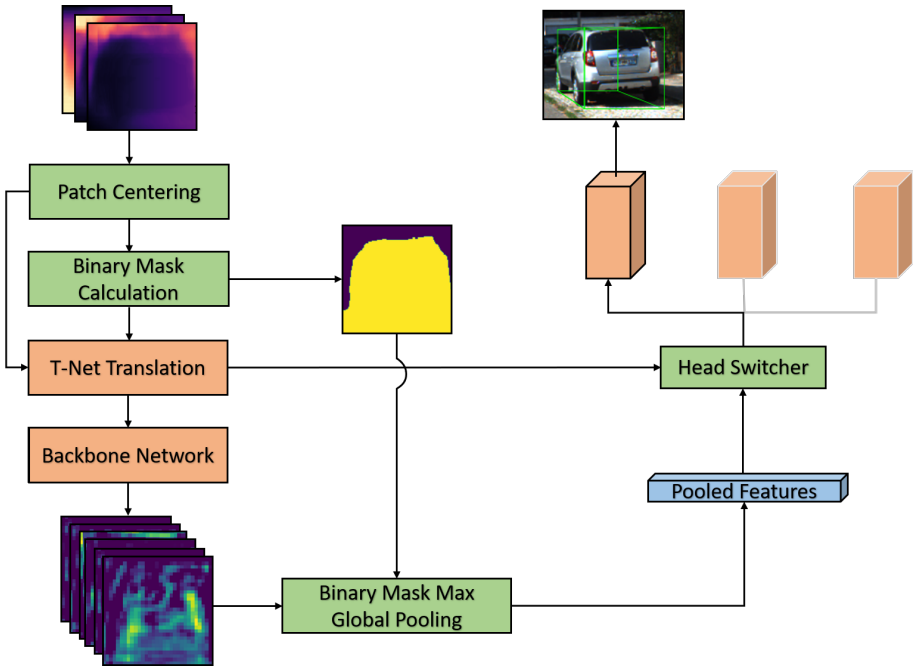


Figure 3.4: Abstraction of the PatchNet 3D Detection Branch. A single patch is processed to output a single 3D bounding box. Images in the illustration are taken from the KITTI train set [1]. Depth estimations were generated using the DORN [12] network.

Before delving into the details of the different modules, it is instructive to describe the core functionality of how the model illustrated in the figure above processes the input patch in order to produce a bounding box. The input patch, which corresponds to a 2D detection, contains pixel-wise x, y and z coordinates. A sequence of standardizing transformations are then applied to the patch, meant to center the object contained in the patch. This centered patch is then used as input to a backbone network (a ResNet-18), which outputs a set of feature maps. These feature maps are then globally max-pooled to form a single feature vector, and used as input to an MLP. The MLP outputs the final bounding box parameters for the given patch. The remainder of this section will explain these different steps in more detail.

In the *Patch Centering* block, the input patch $\mathbf{p}$ is centered by first rotating it around the y-axis such that the mean vector of the patch lies in the yz-plane in the camera frame, and then subtracting the mean position from all pixels,

$$\mathbf{p} \longrightarrow \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} (\mathbf{p} - \langle \mathbf{p} \rangle), \quad \theta = \arctan\left(\frac{\langle x \rangle_{\mathbf{p}}}{\langle z \rangle_{\mathbf{p}}}\right).$$

Note that we do not perform any rotation around the x-axis. Since both the rotation and translation transformation can be calculated directly from a patch, and can be undone via the inverse transform, it is beneficial to apply these transformations

deterministically prior to applying any learnable operations. Otherwise, the backbone network would be required to learn these transformations in order to produce the correct bounding boxes. As such, the purpose of this centering process is to standardize patches, in order to aid the learning process.

After centering, the *Binary Mask Calculation* block generates a mask $\mathbf{M}$ which is intended to exclude background regions in the patch

$$\mathbf{M}(u,v) = \begin{cases} 0, & \mathbf{p}(u,v) > T \\ 1, & \mathbf{p}(u,v) \leq T \end{cases}, \quad T = \langle z \rangle_{\mathbf{p}} + \delta_T.$$

Here, the threshold T for masking a pixel (u,v) is determined by the mean z-value of the centered patch, plus the *threshold offset* δ_T.

The *T-Net Translation* block acts as a second-stage center translation of the patch. T-Net has a lightweight CNN backbone, consisting of three convolutional layers with 128, 128 and 256 kernels of size 1×1, followed by global max pooling and a three-layer MLP with hidden sizes 256, 128, and 3. T-Net takes the masked centered patch as input and outputs an estimate of the 3D bounding box center, to which the patch is subsequently translated:

$$\mathbf{p} \longrightarrow \mathbf{p} - \text{T-Net}\left(\mathbf{M}, \mathbf{p}\right),$$

where $\text{T-Net}\left(\mathbf{M}, \mathbf{p}\right)$ denotes the function implemented by T-Net. The binary mask is used as input to T-Net since it is applied to the feature maps produced by the convolutional backbone before they are max-pooled. As with the previous centering block, the purpose of the T-net is to further standardize the patch in order to aide the learning process, but also to provide an initial depth estimate of the object of interest which is used by the head switcher block.

The resulting patch is then fed through the *Backbone Network*, which is a ResNet-18. After having applied a sequence of non-linearities to the input patch, the backbone outputs a feature tensor F with $d_f = 512$ feature maps, while retaining the pixel dimensions $d_u = d_v = 32$. This preservation of the spatial dimensions is achieved by removing all downsampling layers of the original ResNet (compare with Table 2.1). Each feature map is reduced to a scalar value in the *Binary Mask Max Global Pooling* block, producing a feature vector

$$\mathbf{f}_i = \max_{(u,v)} \mathbf{M}(u,v) \circ F_i(u,v), \quad i \in 1, ..., d_f.$$

In other words, the binary mask is applied element-wise in the pixel-pixel dimension to F. The masked feature maps, $\mathbf{f_i}$, are then max-pooled, resulting in a single output value for each feature map that together make up the *pooled features*. The binary mask is applied to the feature maps to filter out irrelevant background points, and is shown in [19] to have a positive impact on performance. The resulting pooled features make up a deep representation of the input patch.

In the final 3D bounding box regression step, the max-pooled features are processed by an MLP which then outputs the box parameters. We employ the use of multiple

regression heads $H_i(\mathbf{f})$, covering designated distance intervals, i.e., H_1 covers the interval $[0, d_1]$, H_2 covers $[d_1, d_2]$ and so on. This is shown to increase performance in [19]. The *Head Switcher* block ensures that the correct head is chosen with respect to the T-Net center estimate. In other words, given an initial center estimate of the target object, the regression head with the interval covering this estimate is used to regress the bounding box. In a modification to the original PatchNet architecture, each regression head also has an associated confidence estimation head, which outputs confidence scores which reflect the quality of the 3D detection. The addition of this confidence estimation mechanism was shown to have a significant positive impact on performance in [20]. Figure 3.5 illustrates how a single head processes the pooled features.

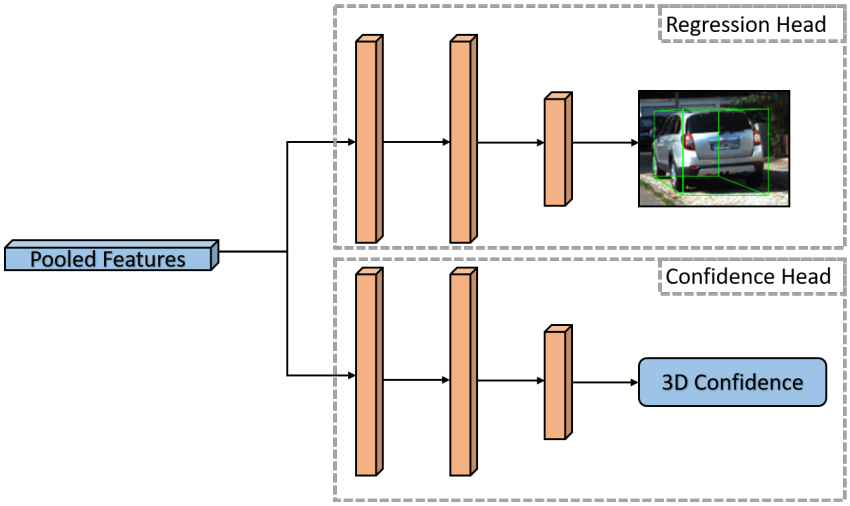


Figure 3.5: Illustration of regression head layout. The head chosen by the head switcher consists of two MLPs, one which regresses the bounding box parameters, and one which outputs a 3D confidence. Images in the illustration are taken from the KITTI train set [1].

In the above figure, the regression- and confidence estimation heads both take the pooled features as input. Both heads consist of MLP with three fully connected hidden layers with 512, 512 and 256 units, respectively (illustrated as vertical blocks). The units of the output layer of each regression head parametrize the bounding box, whereas the single output unit in each confidence head corresponds to the 3D confidence score of the corresponding 3D bounding box prediction.

3.5 Attention Mechanism

The primary contribution of this thesis is that we replace the binary mask in PatchNet, by means of an attention mechanism. In this section, we detail how this binary mask is implemented via a multi-head attention mechanism.

While the binary mask described in Section 3.4.1 is often successful in filtering out background pixels in patches, it also has some drawbacks. As the mask is

generated by tossing all pixels which have a corresponding depth value above the mean depth value of the patch plus a small threshold, it is based on the assumption that objects of interest to a patch are approximately located around this mean depth value. However, this assumption sometimes introduces unwanted information loss. Consider Figure 3.6 as an example.

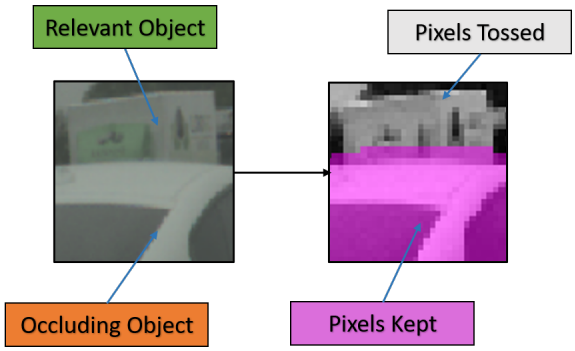


Figure 3.6: Illustration of information loss from using the PatchNet binary mask. The left image shows an occluded object, and the right image shows, in pink, which pixels are selected as relevant by the binary mask. Images in the illustration are taken from the KITTI train set [1].

In the above figure, we see how the object relevant to a patch is heavily occluded by a car in the foreground. The figure shows how the binary mask generated in the PatchNet architecture results in almost all pixels/points belonging to the object of interest are set tossed, while the occluding object remains fully intact. Since occluded objects make up roughly 50% of our data set (see Table 4.2), cases such as the above are a major drawback with PatchNet. A similar effect can also be seen at varying distances. For close-by objects, the depth maps are usually clear and of high quality, resulting in a quite satisfactory performance of the binary mask. However, as distance increases, the depth estimations naturally become more blurry, resulting in much more stochastic binary masks, which induces significant information loss in the patch. As such, it is desirable to replace this binary mask by something more dynamic.

Our proposed attention mechanism replaces replaces the Binary Mask Max Global Pooling block in Figure 3.4 with a Multi-Head attention module. The resulting modified 3D Regression Branch is shown in Figure 3.7.

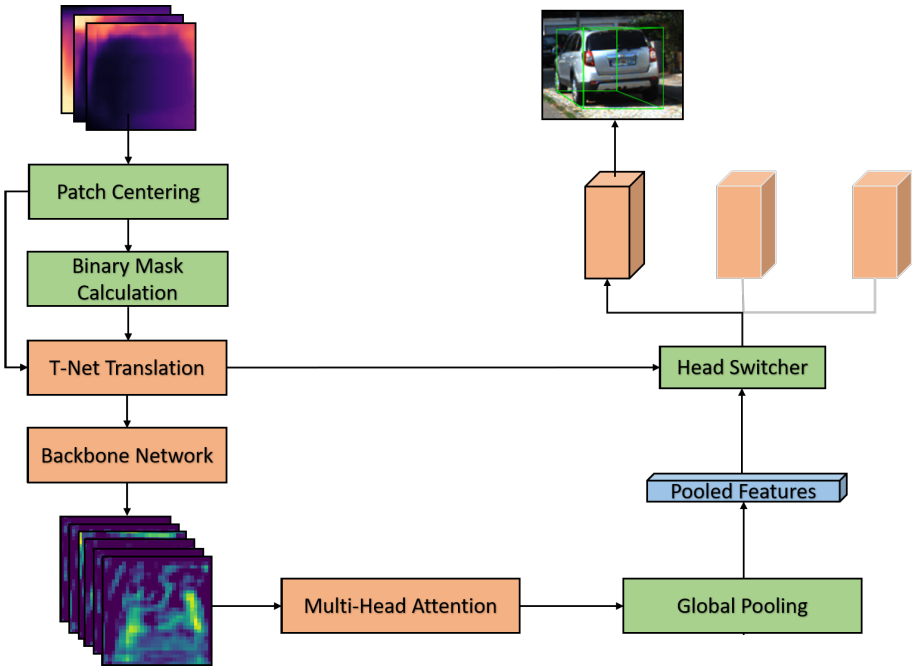


Figure 3.7: Modified 3D Regression Branch architecture. We replace the mask max global pooling operation with our multi-head attention module. Note that the T-Net still uses the binary mask. Images in the illustration are taken from the KITTI train set [1]. Depth estimations were generated using the DORN [12] network.

Inspired by the mechanism in [41] presented in Section 2.3.1, the general idea of our attention module is to use the set of feature maps produced by the ResNet-18 backbone network to construct multiple *attention maps*, acting as a soft mask to attend to certain regions of the feature maps. Each attention map is produced via a single *attention head*. Figure 3.8 illustrates the proposed attention mechanism, detailing how a single attention head operates.

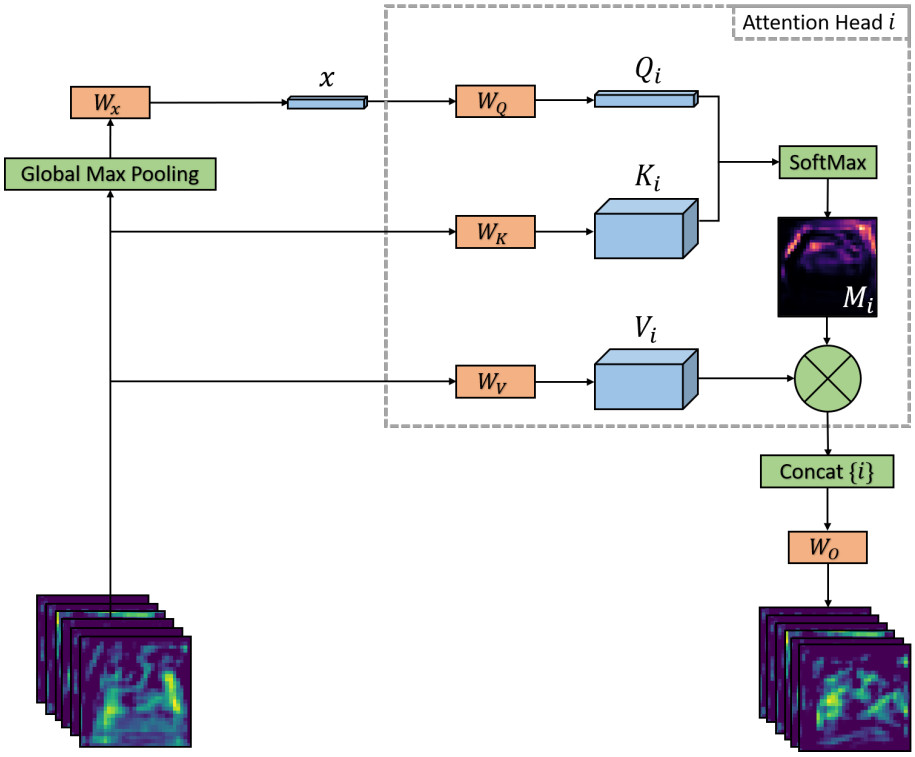


Figure 3.8: Detailed view of the proposed attention mechanism.

At the input of the mechanism are the backbone feature maps $\mathbf{F} \in \mathbb{R}^{d_f \times d_u \times d_v}$. An embedding vector $\mathbf{x}$ is produced from the feature maps by means of global max pooling, followed by multiplication with a trainable weight matrix $\mathbf{W_x} \in \mathbb{R}^{d_f \times d_x}$, such that $\mathbf{x}$ is projected from the embedding dimension d_x to the query dimension d_q. Note that the same embedding is used by *all* attention heads. Based on the embedding vector $\mathbf{x}$ and the ResNet feature maps $\mathbf{f}$, each attention head produces via linear projection a triplet of tensors consisting of a Query, Key and Value tensor. The Query tensor $\mathbf{Q}_i$, Value tensor $\mathbf{V}_i$ and Key tensor $\mathbf{K}_i$ of head $\#i$ are calculated as

$$\begin{cases} \mathbf{Q}_i = \mathbf{W}_Q^i \mathbf{x}, \\ \mathbf{V}_i = \mathbf{W}_V^i \mathbf{F}, \text{ and} \\ \mathbf{K}_i = \mathbf{W}_K^i \mathbf{F}, \end{cases}$$

where the weight matrices

$$\begin{cases} \mathbf{W}_Q^i \in \mathbb{R}^{d_x \times d_Q}, \\ \mathbf{W}_V^i \in \mathbb{R}^{d_f \times d_V}, \text{ and} \\ \mathbf{W}_K^i \in \mathbb{R}^{d_f \times d_K} \end{cases}$$

are comprised of trainable parameters. In the above, d_x, d_f, d_Q, d_V and d_K denote the dimensions of the embedding vector, feature maps, queries, values and keys, respectively. In what follows, we omit the head index i for notational clarity.

Before computing alignments between the query vector and the key tensor, we normalize the query vector

$$\mathbf{Q} \longrightarrow \frac{\mathbf{Q}}{\|\mathbf{Q}\|}$$

where $\|\cdot\|$ denotes the Euclidean norm, in order to reduce the impact of its norm on the final attention mask. In practice, we also note that this generally results in less saturation of the softmax function applied at a later stage.

We then compute alignment scores between the query vector and each pixel in the key tensor, as illustrated in Figure 3.9.

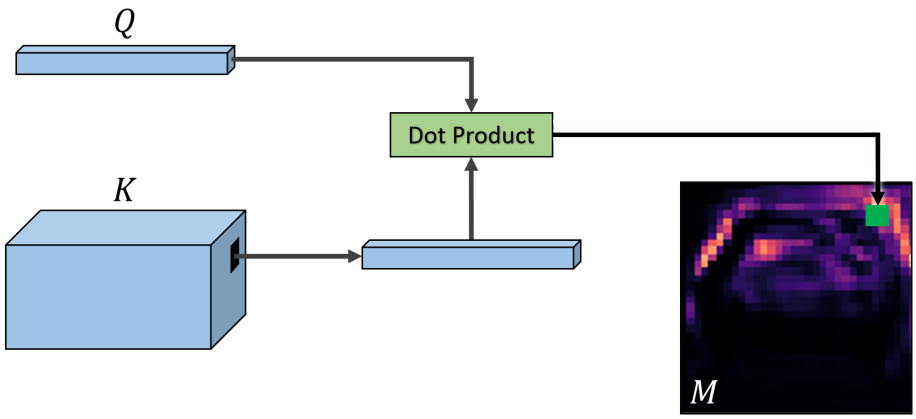


Figure 3.9: Alignment computation between queries and keys in the proposed attention mechanism. The calculated alignment determines the value of a single pixel in the mask, illustrated as a green box here. Note that the softmax operation is not included in this illustration.

In the above figure, we see how each element in the mask matrix $\mathbf{M} \in \mathbb{R}^{d_u \times d_v}$ is computed pixel-wise such that

$$\mathbf{M}_{uv} = \frac{\mathbf{Q}^\top \mathbf{K}_{uv}}{\sqrt{d_k}}.$$

In other words, each pixel value in the mask matrix is equal to the dot-product alignment between the query vector and the corresponding pixel in the key tensor. We then apply the softmax function on $\mathbf{M}$ across the both axes,

$$\mathbf{M} \longrightarrow \text{Softmax}\left(\mathbf{M}\right),$$

thus obtaining a matrix of attention weights.

Having obtained the soft mask $\mathbf{M}$, the next step is to apply these to the corresponding value tensor $\mathbf{V}$. Figure 3.10 shows this process.

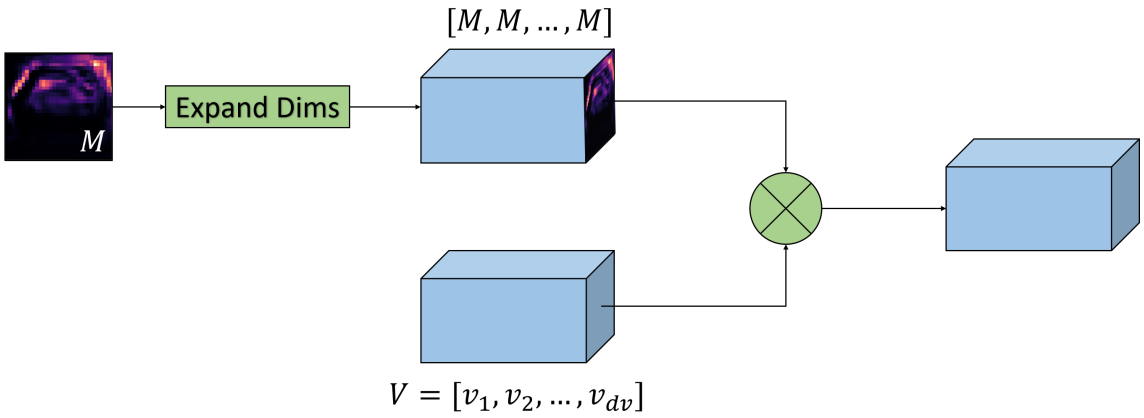


Figure 3.10: Masking procedure in the proposed attention mechanism.

As seen in the above figure, the mask is multiplied element-wise to each feature map of the corresponding value tensor $\mathbf{V}$, such that the output $\mathbf{O}_i$ becomes

$$\mathbf{O} = [\mathbf{M} \circ \mathbf{v}_1, ..., \mathbf{M} \circ \mathbf{v}_{d_v}].$$

In the figure, this is illustrated by a block which expands the feature dimension, such that the mask is repeated across this dimension in order to match the feature dimensionality d_v of the value tensor. Then each such feature map is multiplied point-wise with each corresponding feature map in the value tensor, such that the feature maps in the value tensor are "masked" by the attention mask.

Finally, we concatenate the outputs $\mathbf{O}_i$ from each attention head and linearly project them using a weight matrix $\mathbf{W}_O \in \mathbb{R}^{(N \times d_v) \times d_f}$ in order to project them to the appropriate dimensionality. Note that we have now reintroduced the head index i. As such, the total output $\mathbf{O}$ from the attention mechanism becomes

$$\mathbf{O} = \mathbf{W}_O \left[\mathbf{O}_1, ..., \mathbf{O}_N\right].$$

In theory, what we should have obtained at this point are transformed feature maps, which contain features which are weighted by their relevance to the bounding box prediction, much like how the foreground points are selected by the binary mask.

Much like in PatchNet, we then pool the attended feature maps in order to produce a feature vector, which is used as input to the regression heads. In this thesis, we experiment with two types of pooling:

- **Average pooling**: By summing across the image dimensions of the attended feature maps, the resulting feature vector can be seen as a weighted average-pooled vector, where the weights of the averaging are the attention maps.
- **Max pooling**: By taking the max value of the attended feature maps, we use the same operation as the one employed in PatchNet.

The feature vector produced by the attention module is subsequently processed analogously to the feature vector in PatchNet.

3.6 Loss Function

Choosing a suitable loss function is crucial for training a well functioning model. In this work, we use the loss function used in [19] and [18] for all experiments. This loss incorporates smart heuristics in order to help the model. The total loss function and its components is given by

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{center}} + \mathcal{L}_{\text{yaw}} + \mathcal{L}_{\text{object}} + \mathcal{L}_{\text{corner}} + \mathcal{L}_{\text{confidence}}.$$

The respective components are described in this section. Note that, throughout this section, we neglect weight coefficients for the different components in the total loss for notational clarity. These coefficients will be numerically specified in the following experiments section.

For ease of reading, model outputs are denoted with hats (e.g., $\hat{x}$). For each element in a batch, the only head which will will contribute to the total loss calculation is the head chosen in the Head Switcher block. Since the loss is calculated in the same way for the respective heads, it will be implicitly assumed in this section that we are only considering outputs from the chosen head.

For our size and yaw losses, we use a hybrid classification and regression formulation of $\mathcal{L}_{\text{yaw}}$ and $\mathcal{L}_{\text{object}}$, which has shown to help with model performance in previous work [48, 49]. Moreover, it is employed in both PatchNet [19] and F-PointNets [7]. A combination of regression and classification entails discretizing the target range of values into bins, and performing classification over these bins. As a second step, one regresses a residual value to the center of the classified bin.

3.6.1 Center Loss

Our model produces a preliminary center estimate $\hat{\mathbf{r}}^{\text{T-Net}} = (\hat{x}, \hat{y}, \hat{z})^{\text{T-Net}}$, as well as a final center estimate $\hat{\mathbf{r}} = (\hat{x}, \hat{y}, \hat{z})$. The total center loss contains ℓ_1 loss-terms for the center estimates, both of which are trained towards the ground-truth center $\mathbf{r} = (x, y, z)$

$$\mathcal{L}_{\text{center}} = \sum_i \ell_1 \left(\mathbf{r}_i, \hat{\mathbf{r}}_i\right) + \ell_1 \left(\mathbf{r}_i, \hat{\mathbf{r}}_i^{\text{T-Net}}\right),$$

where $\ell_1 \left(\mathbf{a}, \mathbf{b}\right) = \sum |a_i - b_i|$.

3.6.2 Yaw Loss

In this work, the yaw angle is defined such that $\alpha_{\text{yaw}} \in [-\pi, \pi)$, where an object with $\alpha_{\text{yaw}} = 0$ faces along the positive y-axis in the camera frame. We may also represent angles by placing them in one of n_{yaw} equally sized bins, and recording the residual from the bin center to the original angle, as illustrated in Figure 3.11.

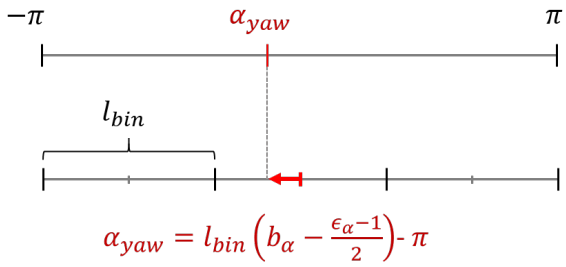


Figure 3.11: Transformation from a continuous angle α_{yaw}, to a bin (vertical red line) and residual (red arrow). In this illustration, $n_{\text{yaw}} = 3$ is used. For yaw angle α_{yaw}, b_α and ϵ_α denote the corresponding bin center and normalized residual.

With n_{yaw} bins, each bin is of length $l_{\text{bin}} = 2\pi/n_{\text{yaw}}$, and the i:th bin center is thus located at $l_{\text{bin}}\left(i - \frac{1}{2}\right) - \pi$ given that i starts at 1. With this notation, a given yaw angle α_{yaw} can be represented as

$$\alpha_{\text{yaw}} = l_{\text{bin}}\left(b_\alpha - \frac{\epsilon_\alpha - 1}{2}\right) - \pi.$$

Here, b_α and ϵ_α are the bin index and normalized residual angle to which the angle is mapped.

Given a patch, this angle representation allows us to separately classify the bin and regress the normalized residual, instead of directly regressing the angle. Each regression head has n_{yaw} softmaxed units in the output layer which represent the probabilities p_i of bin classes b_i, $i = 1, ..., n_{\text{yaw}}$, each with their respective tanh-activated output unit representing the normalized residual ϵ_i. For a transformed ground-truth yaw angle

$$\alpha_{\text{yaw}}^{\text{GT}} \rightarrow \epsilon_\alpha^{\text{GT}}, b_\alpha^{\text{GT}},$$

our total yaw loss is defined as

$$\begin{aligned}\mathcal{L}_{\text{yaw}} &= \mathcal{L}_{\text{yaw}}^{\text{cls}} + \mathcal{L}_{\text{yaw}}^{\text{res}} \\ &= \sum_{i=1}^{n_{\text{yaw}}} \delta_{b_\alpha^{\text{GT}}}^i \left[-\log p_i + \ell_1\left(\epsilon_i, \epsilon_\alpha^{\text{GT}}\right)\right],\end{aligned}$$

comprised of a component for the classification task, and a ℓ_1 component for regressing the residual. Note that the Kronecker delta $\delta_{b_\alpha^{\text{GT}}}^i$ ensures that only unit corresponding to the ground truth bin is trained. However, during evaluation and inference the highest scoring bin and its corresponding residual are used to produce output angle.

3.6.3 Object Loss

The object loss jointly manages object classification and size regression. Given that we have $i = 1, ..., n_{\text{class}}$ classes. The classification loss is once again a categorical cross-entropy loss, computed from n_{class} softmaxed probabilities p_i at the output layer. For regressing the object size, we make use of a list of template sizes (h_i^0, w_i^0, l_i^0)

for the different object classes (chosen such that they reflect the dimensions of a normal car, etc.). At the output layer, $3 \times n_{\text{class}}$ units $(h_i^\delta, w_i^\delta, l_i^\delta)$ with linear activations represent the normalized scale factors to the regressed object sizes

$$\begin{pmatrix} h_i \\ w_i \\ l_i \end{pmatrix} = \begin{pmatrix} h_i^0 \\ w_i^0 \\ l_i^0 \end{pmatrix} \circ \begin{pmatrix} h_i^\delta \\ w_i^\delta \\ l_i^\delta \end{pmatrix}. \tag{3.1}$$

Only the size regressed for the true class c_{GT} is considered in the loss during training, and as such, the total object loss becomes

$$\begin{aligned} \mathcal{L}_{\text{object}} &= \mathcal{L}_{\text{object}}^{\text{cls}} + \mathcal{L}_{\text{object}}^{\text{res}} \\ &= \sum_{i=1}^{n_{\text{class}}} \delta_{c_{\text{GT}}}^i \left[-\log p_i + \left\| \begin{pmatrix} h_{\text{GT}} - h_i \\ w_{\text{GT}} - w_i \\ l_{\text{GT}} - l_i \end{pmatrix} \right\|_{\ell_1} \right], \end{aligned}$$

Note that during inference and validation, the output size (h, w, l) will be determined by Equation 3.1 with i chosen such that p_i is maximized.

3.6.4 Corner Loss

First proposed in F-PointNets [7], the corner loss aims to regularize, and keep track of the overarching task of regressing a 3D bounding box, such that none of the other sub-tasks dominate the total loss. The corner loss extracts the output center position, yaw angle and box size in in order to compute positions of the box corners. With the notation introduced in Figure 2.8, we can write down the corner loss as

$$\begin{gathered} \mathcal{L}_{\text{corner}} = \min \left\{ \sum_{ijk} \left\| \mathbf{r}_{ijk} - \mathbf{r}_{ijk}^{\text{GT}} \right\|_{\ell_1}, \sum_{ijk} \left\| \mathbf{r}_{ijk}^{\text{flip}} - \mathbf{r}_{ijk}^{\text{GT}} \right\|_{\ell_1} \right\}, \\ i \in \{U, D\}, \ j \in \{L, R\}, \ k \in \{F, B\}. \end{gathered}$$

Here, we sum over all ℓ_1 norms of difference vectors between the regression- and ground-truth box corners. Note that we also compute this error for a box $\mathbf{r}_{ijk}^{\text{flip}}$, which has had its direction flipped, such that the yaw angle does not cause abnormal loss spikes if flipped compared to the ground truth.

3.6.5 Confidence Loss

As illustrated in Figure 3.5, we use a confidence estimation head for each respective regression head, as proposed in [20]. The motivation for this is that using the class-confidences stemming from the 2DOD step as 3D detection confidences, as is common in set-ups such as this, very often results in over-confident detections, since the 2D class-confidences lack any real connection to the actual task performed by the 3D Detection Branch. The confidence head chosen with respect to the head switcher outputs a confidence C^{3D}, and the corresponding loss is computed as

$$\mathcal{L}_{\text{conf}} = -T \log C^{\text{3D}} - (1 - T) \log \left(1 - C^{\text{3D}}\right), \quad T = \exp\left[-\beta \mathcal{L}_{\text{corner}}\right].$$

Here, $\beta > 0$ is a temperature parameter, and $\mathcal{L}_{\text{corner}}$ is the corner loss. By interpreting T as a probability, and by noting that the corner-loss likely reflects how good a given 3D bounding box is, the confidence heads will in theory learn to output a value $\in [0, 1]$ which measures the quality of a detection, and can thus be interpreted as a dedicated 3D confidence.

4

Experiments

In this chapter, we describe the experimental set-ups used to evaluate the architectures described in Chapter 3.

4.1 Data

Our base data set consists of roughly 160k images, which we split into a training set consisting of 75% of the images and validation set consisting of 25% of the images. This split is performed with respect to the date when the images were taken, in order to reduce the risk of a geographical overlap like the one discovered in [20] for the KITTI data set. The split used for the baseline provided by Zenseact does not contain a test set. As such, we solely evaluate our models on the validation set, since there is no baseline performance to compare with (due to confidentiality agreements, we must report relative metric values). The implications of this delimitation are addressed in more detail in Section 6.2.

Each image contains objects with classes cars, vans and trucks, with corresponding annotations which contain ground truth information about the type of object, their 2D and 3D bounding boxes, as well as level of occlusion. The 3D bounding boxes are parametrized as specified in section 2.4.2. The data is supplied and annotated by Zenseact, and due to confidentiality we will not describe the data set in more detail.

Using the base data set, we produce a set of patches. This is done by running inference on the images using both the mono-depth network and the 2DOD network, obtaining a set of true positive 2D detections with respect to a 2D IoU-threshold of 0.5. Using the depth map, and the obtained 2D bounding boxes, all computations up until the 3D Regression Branch are done prior to training (see Figure 3.1), such that the set of patches constitute the actual data used for the training of the 3D Regression Branch. We also want to emphasize that neither the mono-depth or 2D object detection network use validation examples in their respective training sets.

Since our 2DOD network also outputs RGB-based 3D detection boxes, we also obtain a purely image-based 3D detection for each patch. Due to confidentiality agreements, we cannot describe the 3D detector in more detail. In subsequent sections, we will refer to this model simply as “Baseline”.

4.1.1 Data Set Filtering

In order to evaluate models in different settings, we filter the data set on distance and occlusion level, inspired by the difficulty categories of the KITTI data set [1]. Following the distance intervals used in [19], we also append an interval for the very faraway objects. As such, we define the following distance levels in Table 4.1.

`distance categories`	Distance	% of patches
Near	0 - 30 m	21.4%
Mid	30 - 50 m	17.1 %
Far	50 - 100 m	29.5%
Very far	>100 m	32.0%

Table 4.1: Distance categories.

Table 4.2 shows how the data is separated into different occlusion levels. When filtering the data on occlusion, we restrict the distance to 0 - 100 m. This is due to the fact that highly occluded objects at greater distances are extremely difficult to detect, and add significant noise to the metric if included. This will be implicit in the following sections.

`occlusion categories`	Occlusion	% of patches
None	0.0%	47.9%
Low	0.01% - 21.0%	23.7%
Medium	21.0% - 51.0%	15.5%
High	51.0% - 100.0%	12.9%

Table 4.2: Occlusion categories.

4.2 Implementation Details

Given our method of producing the entire set of patches preemptively, our experiments are set up such that we only train the 3D Regression Branch of the pipeline. In our experiments, we train and evaluate our models on vehicle-type objects, while using the Adam optimizer with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ for 100 epochs and using a batch size of 64 patches. We use the learning rate scheduling illustrated in Figure 4.1. The initial learning rate is $\eta = 10^{-3}$. Every 40 epochs the learning rate decays by a factor of 10, finishing training with a learning rate $\eta = 10^{-5}$ after 100 epochs.

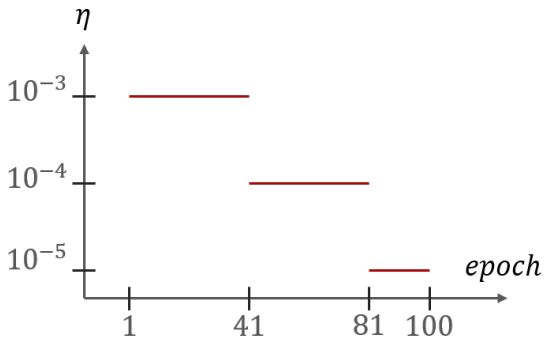


Figure 4.1: Illustration of learning rate scheduler.

Experiments which are conducted in this thesis utilize a total loss function during training which has different weights for the different loss-components,

$$\begin{aligned}\mathcal{L}_{\text{total}} = 1 \cdot \mathcal{L}_{\text{center}} \\ + 1 \cdot \mathcal{L}^{\text{cls}}_{\text{yaw}} + 10 \cdot \mathcal{L}^{\text{reg}}_{\text{yaw}} \\ + 1 \cdot \mathcal{L}^{\text{cls}}_{\text{object}} + 10 \cdot \mathcal{L}^{\text{reg}}_{\text{object}} \\ + 10 \cdot \mathcal{L}_{\text{corner}} \\ + 1 \cdot \mathcal{L}_{\text{confidence}}.\end{aligned}$$

These weights are chosen according to a (non-exhaustive) hyperparameter search, with the goal of finding a stable training process, where all losses contribute somewhat equally to produce good performance. Note that for this specific configuration, the dominating loss is the corner loss, which is based on all the other loss components. Furthermore, we set the confidence temperature parameter β to 10, since it indicated stable performance during preliminary experimentation.

We use this general method of training both for our PatchNet-implementation, and for the model with the incorporated attention mechanism. All models are implemented in Python using the TensorFlow [50] framework.

4.2.1 Regularization

In order to mitigate the effects of overfitting, we use random cropping to augment our training data. When collecting the data set, the patches are padded, such that the random cropping does not crop away significant parts of the objects of interest. During training, each input patch is randomly cropped by 10% along both height and width axes, and then resized to 32×32 before being passed to the 3D regression branch. For evaluation, the random cropping is replaced by center-cropping of the same size. In addition to random cropping, we also randomly offset the entire patch along the z-axis by a small value, randomly sampled from a normal distribution. Furthermore, we employ the use of batch-normalization layers throughout the model.

4.2.2 Evaluation

We evaluate models using the $AP|_{R_{40}}$ metric, and only on vehicle-type objects, with respect to two metric thresholds: bird's eye view IoU with threshold set to 0.5, and RCE with threshold set to 0.05. IoU-based metrics are in standard use object detection model evaluation, but RCE-based metrics are not as common. We choose to employ RCE to better evaluate models in settings where a high IoU might be very difficult to achieve, but where there could still be value in evaluating how close bounding box predictions are, such as for high distances. Figure 4.2 illustrates some examples of true- and false positives, using these thresholds.

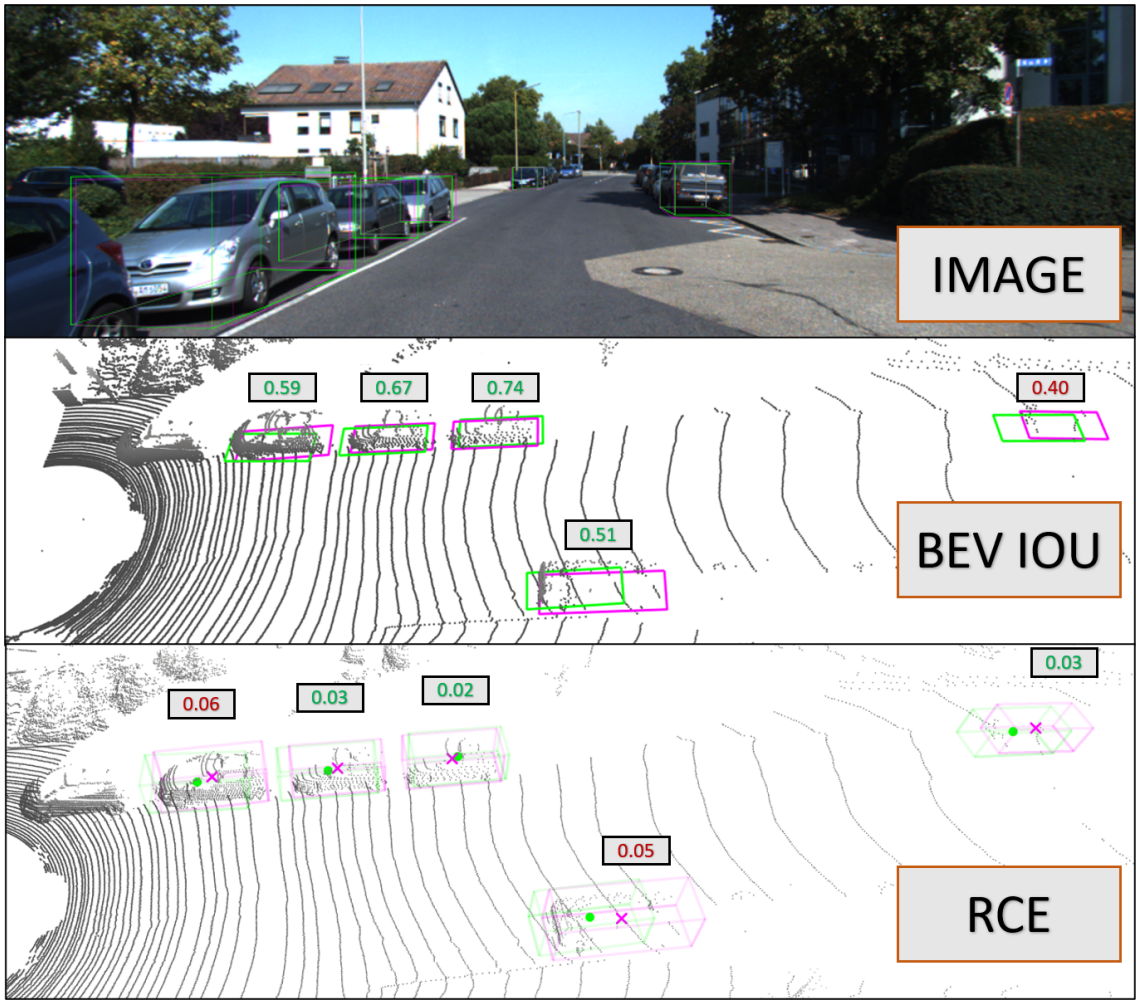


Figure 4.2: The top image illustrates detections (lime) and ground truths (magenta) in the image plane. In the middle image, BEV projections of detections and ground truths are plotted and complemented by the LiDAR point cloud, and each box is annotated with its respective BEV IoU. True and false positives (BEV IoU > 0.5) have been marked with green and red text, respectively. The bottom image highlights the detection- and ground truth centers, with annotations indicating each respective RCE. True positives are detections with RCE < 0.05. Images in the illustration are taken from the KITTI train set [1].

Observe for the RCE, that the absolute center error is allowed to be larger if the GT is further away from the ego-car. This can be clearly understood by considering the left-, and right most detections in the bottom image. As such, it provides a good metric for cars located at high distances, where there might still be value in evaluating the quality of a detection despite it not achieving a high IoU value. Note also that the RCE metric is more restrictive at closer distances, where a high IoU is easier to achieve, showing how the metrics complement each other.

Our metric thresholds are chosen such that they provide an appropriate compromise between being informative and stable. For example, employing a high IoU-threshold will result in a metric which is very sensitive to the slightest changes in output bounding boxes. Conversely, employing a low IoU threshold will result in a metric which is easy to satisfy, and thus provides less discriminative information about the performance of good versus bad models, but is much less sensitive to change in output, thus resulting in a more stable metric.

4.3 Ablation Study Methodology

In order to evaluate how the different models perform in different settings and while using various hyperparameters, we base our experiments around a number of *ablation studies*. These ablation studies are conducted by adjusting one component or hyperparameter in the model at a time. While not strictly in line with the traditional definition of an ablation study in the case of all experiments, our set-up still allows us to systematically investigate the effect of one hyperparameter at a time, instead of navigating the entire high-dimensional hyperparameter-space simultaneously.

For each experiment in an ablation study, we train three separate models, each using identical hyperparameters, but with different random seeds. For each such training, which we will refer to as *runs*, we choose the model from the epoch with the lowest total validation loss, and calculate average precision for each metric by using the inferred set of 3D bounding boxes on the validation set. Finally, we report the estimated mean $\hat{\mu}$,

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i. \tag{4.1}$$

Here, $n = 3$ is the number of runs, while X_i is the observed metric value for the i:th run. Additionally, Equation 4.2 shows how we calculate an unbiased estimate of the standard deviation,

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \hat{\mu})^2}, \tag{4.2}$$

where we divide by $n-1$ instead of n, to account for the fact one degree of freedom is lost when estimating the mean according to Equation 4.1. Performance will be quantified as $\hat{\mu} \pm \hat{\sigma}$ throughout the results section, i.e., the estimated mean with an error-margin corresponding to plus/minus the standard deviation.

In the following sub-sections, we describe the specific ablation studies performed for this thesis. We summarize the experiments in tabular form, naming each respective

experiment such that the reader can quickly jump between here and the results section. Note that the union of all ablation studies contain redundancies, where a specific experiment-set appears in several studies. Such experiments will be reused in the results section, yet still tabulated together for ease of comparison.

4.4 PatchNet Ablation Studies

In this section, we describe the ablation studies performed using the PatchNet architecture, described in Section 3.4.1. Throughout the different ablation studies, we clearly state what components are being varied, and describe why the experiment is of interest, and how it aligns with the research questions.

4.4.1 Distance and Regression Heads Ablations

Since image-based depth estimations exhibit errors which scale drastically with longer ranges [18], we perform a *distance ablation*-study. In this ablation, we study the impact on model performance by inclusion of patches at increasing distances. In other words, for each distance category in Table 4.1 we perform an experiment by training models using training data with ground-truth annotation centers up to and including the distances of that distance category. Table 4.3 illustrates the different experiments we conduct.

distance abl.		included distances
`experiments`	PDA Near	$[0, 30]$
	PDA Mid	$[0, 50]$
	PDA Far	$[0, 100]$
	PDA Very Far	$[0, \infty)$

Table 4.3: *Distance ablation* experiment, performed using model PatchNet with only one regression head, while varying the span of distances in the training set.

For this ablation, the PatchNet architecture is used, using one regression head. Note that while we train some of the models on subsets of the data set, we infer and present results on the entire validation set. This is to investigate the potential generalization capability of a model which is trained on only a subset of the total data set.

The main motivation for this experiment is to study how the feasibility of utilizing mono-depth estimations changes with increased distances. Since the error of mono-depth estimation naturally increases with distance, it is very important to ascertain the upper limits to the feasibility of basing a 3DOD model on such estimations. In order to further isolate the effects of the object distances, we limit the number of regression heads of the model to one. An alternative approach would be to successively add a regression head for each added distance category, but this would

decrease the clarity of the results, as it would not be known if performance difference arise due to change in training data or model complexity.

As noted in Chapter 3, our PatchNet may utilize multiple regression heads, each covering a designated set of distances. In our *regression heads ablation*, we study the impact of incorporating a varying number of regression heads. Table 4.4 illustrates the different experiments we conduct.

		covered distances		
input abl.		first head	second head	third head
experiments	PRHA One	$[0, 100]$	-	-
	PRHA Two	$[0, 50)$	$[50, 100]$	-
	PRHA Three	$[0, 30)$	$[30, 50)$	$[50, 100]$

Table 4.4: Table illustrating the regression heads ablation experiments. The model used is the baseline model, trained on examples within 100 m.

The motivation for this ablation study is to complement the previous ablation study, and to investigate the effects of adding additional regression heads, while keeping the training distances constant. As such, we can evaluate whether the observation made in [19], that using multiple regression heads increases model performance, holds true in the case of our data set.

4.4.2 PatchNet Input Ablation

We perform an *Input Ablation* study to assess whether uncertainty estimates from the mono-depth estimator, and/or RGB data can be utilized to increase the performance of PatchNet. As such, it is a direct investigation of our second research question regarding input data. Table 4.5 illustrates the experiments we conduct. When supplying additional input channels, these channels are simply appended as

		included channels		
input abl.		cartesian	RGB	uncertainty
experiments	PIA Base	✓	-	-
	PIA RGB	✓	✓	-
	PIA Unc	✓	-	✓
	PIA RGB+Unc	✓	✓	✓

Table 4.5: Table illustrating the input ablation experiments. Each row indicates an experiment, and the respective inputs which are used.

extra data channels in addition to the x, y and z channels of the patches. They subsequently undergo many of the same augmentations, including random cropping, and the RGB data is also augmented by small random fluctuations to the hue and saturation, after which each color-channel is re-scaled from $\in [0, 255]$ to $\in [-1, 1]$.

The idea behind this ablation study is that additional input channels might aid the network in producing more accurate predictions in settings where the depth estimates contain little information. For example, RGB-data might provide the network with additional information regarding the type, size and rotation of an object in cases where the depth map has no discernible details. The uncertainty values may provide the network with additional information regarding the quality of the depth estimations.

4.5 Attention Mechanism Ablation Studies

In this section we describe the different experiments we conduct in order to evaluate the effects of replacing the binary mask in PatchNet with our attention mechanism. In all experiments except those belonging to the *attention pooling ablation*, the *Global Pooling* block in Figure 3.7 performs global average pooling. Furthermore, these experiments filter out any training examples further away than 100 m, and use three regression heads with the distance splits described in Table 4.4.

While the goal of the previous ablation studies was to investigate model performance with respect to different settings and the potential performance benefits of including additional input data, the attention ablation studies are, in part, intended as a non-exhaustive hyperparameter search. In other words, the ablation studies related to the attention mechanisms serve the purpose of investigating how different aspects of the attention mechanism influence the performance of the network, in order to arrive at a well-performing set-up. The best performing set-ups are then highlighted in Section 5.3, and discussed in more detail with regards to the research questions.

Based on the results of each ablation study, we choose one set-up to proceed to the subsequent ablations. When choosing this set-up, we take into consideration the observed performance as well as model complexities.

4.5.1 Attention Heads Ablation

Since our attention mechanism allows for an arbitrary number attention heads, we perform an ablation study on how the number of such heads affects performance. Table 4.6 shows the experiments included in this ablation study.

attention head abl.		hyperparameters att. heads	d_x	d_Q	d_K	d_V	# params [10^6]
	PatchNet	-	-	-	-	-	3.9
	AHA 1H	1	512	512	512	512	5.3
experiments	AHA 2H	2	512	512	512	512	6.3
	AHA 4H	4	512	512	512	512	8.4
	AHA 8H	8	512	512	512	512	12.6

Table 4.6: *Attention heads ablation* experiments. Rows describe experiments corresponding to the model described in Section 3.5 with the tabulated dimensionalities. In the rightmost column, the number of trainable parameters of 3D Regression Branch of each model is shown (PatchNet included for comparison).

In the above table, we also included PatchNet, as well as the number of parameters in the 3D Regression Branch of each respective model (i.e., those parameters which are trained for this thesis), so that the reader can understand the how this number is affected when using a varying number of attention heads.

The motivation for this study is to investigate the effect of additional attention heads on the performance of the model in different scenarios. While the binary mask utilized by PatchNet only allows for one mask, including more attention heads may allow the network to simultaneously attend to and specialize in several different parts of the patch. For example, each head in an attention mechanism may learn to specialize in one specific part of the input patch, such as the wheels of the vehicle or perhaps the ground surrounding the object.

4.5.2 Attention Dimensionality Ablation

This ablation study extends the previous by also varying the key, query and value dimensionalities; d_Q, d_K and d_V. Technically, this is an improper use of the word ablation, since we vary several components at once, but we keep the naming convention in order to align with other names. We conduct experiments using four attention heads, shown in Table 4.7.

attention dimensionality abl.		hyperparameters att. heads	d_x	d_Q	d_K	d_V	# params [10^6]
experiments	PatchNet	-	-	-	-	-	3.9
	ADA 4H512×512	4	512	512	512	512	8.4
	ADA 4H512×256	4	512	256	256	256	6.3
	ADA 4H256×128	4	512	128	128	128	5.3
	ADA 4H128×64	4	128	64	64	64	4.5

Table 4.7: *Attention dimensionality ablation* experiments. Rows describe experiments corresponding to the model described in Section 3.5 with the tabulated dimensionalities. In the rightmost column, the number of trainable parameters of the 3D Regression Branch of each model is shown (PatchNet included for comparison).

The motivation behind this experiment is to systematically evaluate how the dimensionality of the attention mechanism affects the performance of the network. While the dimensionality of the components of each attention head may increase its representation power, it also greatly impacts the total parameter count of the network. As such the goal of this ablation is to identify a set-up which compromises between increased model performance and complexity.

4.5.3 Attention Pooling Ablation

This ablation study is conducted to provide quantitative grounds for which pooling operation should be used in the *Global Pooling* block in Figure 3.7. We compare global average- and max pooling, by means of the experiments tabulated in Table 4.8.

attention dim. abl.		hyperparameters att. heads	d_x	d_Q	d_K	d_V	pooling type
experiments	APA MP4	4	512	128	128	128	max
	APA AP4	4	512	128	128	128	avg.

Table 4.8: *Attention pooling ablation* experiments. Rows describe experiments corresponding to the model described in Section 3.5 with the tabulated parameters.

In the PatchNet architecture, a pooling operation is performed on the set of feature maps produced by the backbone CNN from an input patch, resulting in a feature vector. This feature vector is then processed by one of the regression heads which estimates the 3D bounding box parameters.

Our attention mechanism is designed such that it can form soft masks which highlight different features of a patch, such as the wheels of a car. As such, it seems natural to use average pooling, allowing all pixels with non-zero weight to contribute to the features through the mask-weighted average. However, since we base

our model on the PatchNet architecture, it is important to highlight that PatchNet uses max pooling. In [19], it is shown that max pooling yields improved performance over average pooling for the PatchNet architecture. As such, it is of interest to investigate which pooling operation is ideal when using the proposed attention mechanism.

4.5.4 Attention Input Ablation

This ablation study experiments with different input combinations while, in contrast to the previous input ablation, also incorporating the attention mechanism. In this ablation study, we utilize an attention mechanism with average pooling, four heads and dimensionalities $d_x, d_q, d_k, d_v = 512, 128, 128, 128$. We then vary the data used as input to the model backbone, similarly to the previous input ablation study. Table 4.9 shows the experiments performed for this study.

		`included channels`		
attention input abl.		cartesian	RGB	uncertainty
`experiments`	AIA Base	✓	-	-
	AIA RGB	✓	✓	-
	AIA Unc	✓	-	✓
	AIA RGB+Unc	✓	✓	✓

Table 4.9: Table illustrating the input ablation experiments. Each row indicates an experiment, and the respective inputs which are used.

The purpose of this study is to investigate whether the attention mechanism can extract additional information by utilizing additional input channels. We hypothesize that the additional channels might allow the network to more easily attend to the relevant portions of an input patch. For example, if an object is heavily occluded, it might be difficult to identify which pixels belong to that object from the depth map. However, the occluded object might be more clearly visible in the RGB patch. In principle, this could allow the model to produce attention maps, which selectively put high weight values on pixels in the depth map which are seen to belong to the object of interest in the RGB patch. Similarly, uncertainty estimates may be leveraged by the attention mechanism to focus on regions of a patch in which the depth estimator is more certain of its estimates.

5

Results

In this chapter, we provide the results of the studies described in Chapter 4. In addition to describing the results, subsequent sections also include some preliminary discussion and conclusions.

5.1 PatchNet Ablation Studies

In this section we report the experimental results for experiments described in Section 4.4. We present all results in tabular form, quantifying performance using the mean and standard deviation taken over the runs in the experiment, calculated according to equations 4.1 and 4.2. We also remind the reader that, due to confidentiality agreements, we report all results relative to the Baseline, which does not utilize depth estimates.

When inspecting the results, we are tasked with comparing results presented on the form $\hat{\mu} \pm \hat{\sigma}$, calculated as described in the above paragraph. Say for example that results two experiments, a and b, with their performances quantitatively expressed as

$$\texttt{performance of } a = \hat{\mu}_a \pm \hat{\sigma}_a,$$
$$\texttt{performance of } b = \hat{\mu}_b \pm \hat{\sigma}_b.$$

Throughout this section, we say that a performs *significantly* better than b if

$$\hat{\mu}_a - \hat{\sigma}_a > \hat{\mu}_b + \hat{\sigma}_b,$$

i.e., when a has a higher metric value than b and the error intervals do not overlap. While this does not correspond to a formal statistical test, it should give a clear indication that a performs better than b, even when accounting for the variability of the model performance across runs in the respective experiments.

5.1.1 Distance Ablation

Tables 5.1 and 5.2 shows experimental results from the distance ablation study, described in table 4.3.

	$AP\|_{40}$@ BEV IoU 0.5			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.30	0.08	**0.02**
PDA Near	**1.44** ± 0.01	0.30 ± 0.07	0.00 ± 0.00	0.00 ± 0.00
PDA Mid	1.43 ± 0.04	0.49 ± 0.03	0.01 ± 0.00	0.00 ± 0.00
PDA Far	1.36 ± 0.09	**0.49** ± 0.03	**0.10** ± 0.01	0.00 ± 0.00
PDA Very Far	1.29 ± 0.03	0.49 ± 0.02	0.10 ± 0.00	0.01 ± 0.00

Table 5.1: $AP|_{40}$@ BEV IoU 0.5 for PDA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

	$AP\|_{40}$@ RCE, threshold 0.05			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.92	0.73	**0.72**
PDA Near	1.61 ± 0.03	0.81 ± 0.15	0.01 ± 0.01	0.00 ± 0.00
PDA Mid	**1.62** ± 0.05	**1.29** ± 0.03	0.11 ± 0.02	0.00 ± 0.00
PDA Far	1.50 ± 0.14	1.25 ± 0.03	**0.83** ± 0.04	0.02 ± 0.00
PDA Very Far	1.38 ± 0.06	1.22 ± 0.00	0.79 ± 0.01	0.45 ± 0.01

Table 5.2: $AP|_{40}$@ RCE, threshold 0.05 for PDA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

In the above tables, we have divided all elements by the metric value obtained by the baseline model in the nearest distance category. We want to once again remind the reader that, here, "Baseline" refers to the single shot, purely image based joint 2DOD and 3DOD model. The normalization is conducted due to confidentiality agreements. Subsequent sections will also perform this type of normalization.

From tables 5.1 and 5.2, we observe a few clear trends regarding model performance across different distances. First and foremost, the mono-depth based models perform significantly better in comparison to the baseline model in the categories on which they have trained. The clear exception to this is the Very Far category, where all mono-depth models are outperformed by the baseline model.

A general trend for all models and metrics, including the baseline, where performance in terms of both metrics decrease with increased object distance. This is especially true for the BEV IoU metric, as exemplified by the ten-fold drop in performance when considering objects in the near category as opposed to the far category. The reason for this drop can, in part, be understood by considering Figure 4.2, where it is evident that the BEV metric is much more unforgiving at greater distances. The fact that the drop in the RCE metric is not as harsh further highlights that it is useful to have multiple metrics, capturing different aspects of model performance.

We clearly observe that models do not generalize well outside the distance intervals on which they have been trained, a trend seen in both metrics. To see this, we can compare the performances of PDA Near and PDA Far. PDA Near performs very well compared to the baseline model in the Near category, which is the category on which it was trained. However, in the following distance categories, PDA Near performs considerably worse, especially in the far and very far category. In contrast, PDA Far, which has trained on all categories up to and including the Far category, performs better than baseline in all categories on which it has trained.

We observe that as higher distances are included in the training set, lowered performance is observed in lower distance categories. In other words, there is a negative trend in performances in e.g., the Near category in terms of both metrics as longer distance intervals are included in the training set. For example, PDA Very Far performs significantly worse, w.r.t. the mean and error margins, than PDA Near and PDA Mid in the Near category, even though it has trained on a data set which contains the data set on which PDA Near and Far have trained on. This trend is also seen in the Mid and Far categories in terms of the RCE metric.

From the distance ablation study, we conclude that it is reasonable to exclude training examples that belong to the Very Far category, as these training examples seem to degrade the overall performance for closer distances, while simultaneously observing that PatchNet is not quantitatively competitive with the Baseline model.

5.1.2 Regression Heads Ablation

Table 5.3 and 5.4 show experimental results from the regression head ablation study, described in table 4.4. Keep in mind that the training set used for these models does not include the Very Far distance category.

	$AP\|_{40}$@ BEV IoU 0.5			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.30	0.08	**0.02**
PRHA One	1.36 ± 0.09	<u>0.49</u> ± 0.03	0.10 ± 0.01	<u>0.00</u> ± 0.00
PRHA Two	<u>1.40</u> ± 0.01	0.49 ± 0.01	<u>0.10</u> ± 0.01	0.00 ± 0.00
PRHA Three	**1.41** ± 0.02	**0.50** ± 0.01	**0.10** ± 0.00	0.00 ± 0.00

Table 5.3: $AP|_{40}$@ BEV IoU 0.5 for PRHA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

	$AP\|_{40}$@ RCE, threshold 0.05			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.92	0.73	**0.72**
PRHA One	1.50 ± 0.14	**1.25** ± 0.03	<u>0.83</u> ± 0.04	<u>0.02</u> ± 0.00
PRHA Two	<u>1.52</u> ± 0.03	1.22 ± 0.01	0.83 ± 0.05	0.02 ± 0.01
PRHA Three	**1.53** ± 0.07	<u>1.22</u> ± 0.02	**0.84** ± 0.01	0.02 ± 0.00

Table 5.4: $AP|_{40}$@ RCE, threshold 0.05 for PRHA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

Tables 5.3 and 5.4 do not imply that there is any significant performance gain when including multiple regression heads, and no clear trend is observed as the number of heads is increased, which is contrary to the result observed in [19]. However, the average overall performance of PRHA Three is slightly above the other experiments. Furthermore, PRHA three seems to exhibit slightly more stable performance across different runs in the Mid and Far distance categories.

5.1.3 PatchNet Input Ablation

Table 5.5 and 5.6 show the experimental results from the input ablation study, described in table 4.9.

	$AP\|_{40}$@ BEV IoU 0.5			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.30	0.08	**0.02**
PatchNet	1.41 ± 0.02	0.50 ± 0.01	0.10 ± 0.00	<u>0.00</u> ± 0.00
PIA RGB	**1.42** ± 0.00	<u>0.52</u> ± 0.01	**0.12** ± 0.01	0.00 ± 0.00
PIA Unc.	<u>1.41</u> ± 0.01	0.49 ± 0.02	0.11 ± 0.01	0.00 ± 0.00
PIA RGB+Unc	1.39 ± 0.03	**0.55** ± 0.03	<u>0.12</u> ± 0.02	0.00 ± 0.00

Table 5.5: $AP|_{40}$@ BEV IoU 0.5 for PIA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

	$AP\|_{40}$@ RCE, threshold 0.05			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.92	0.73	**0.72**
PatchNet	1.53 ± 0.07	1.22 ± 0.02	0.84 ± 0.01	0.02 ± 0.00
PIA RGB	**1.58** ± 0.03	**1.30** ± 0.05	**0.93** ± 0.06	0.04 ± 0.00
PIA Unc.	1.58 ± 0.01	1.24 ± 0.01	0.86 ± 0.02	0.01 ± 0.01
PIA RGB+Unc	1.55 ± 0.06	1.29 ± 0.05	0.88 ± 0.08	0.03 ± 0.02

Table 5.6: $AP|_{40}$@ RCE, threshold 0.05 for PIA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

From tables 5.6 and 5.5, we observe that including additional input channels does have an impact on performance. Firstly, including the RGB channels yields significant performance increases in the Mid and Far setting, for both metrics (PIA RGB). Secondly, including aleatoric uncertainty estimates loses/gains 1% in the Mid/Far categories as compared to the Baseline. Finally, incorporating both RGB and uncertainty (PIA RGB+Unc) does not yield a significant advantage over PIA RGB.

The results show that the common denominator for the most significant increases in performance is the addition of the RGB channels. These performance increases are especially apparent in the Mid and Far distance intervals, while the significance of these improvements is weak in the Near category.

Table 5.7 shows results for different levels of occlusion in the BEV setting.

	$AP\|_{40}$@ BEV IoU 0.5			
Occlusion	No	Low	Med	High
Baseline	1.00	0.37	0.24	0.05
PatchNet	1.90 ± 0.02	0.77 ± 0.02	**0.42** ± 0.02	**0.08** ± 0.01
PIA RGB	1.95 ± 0.06	**0.79** ± 0.02	0.40 ± 0.05	0.06 ± 0.00
PIA Unc.	1.93 ± 0.04	0.77 ± 0.02	0.39 ± 0.01	0.07 ± 0.01
PIA RGB+Unc	**1.96** ± 0.05	0.78 ± 0.05	0.39 ± 0.04	0.06 ± 0.01

Table 5.7: $AP|_{40}$@ BEV IoU 0.5 for PIA experiments, split by occlusion. Tabulated values are all divided by the baseline $AP|_{40}$ score in the No occlusion category, due to confidentiality reasons.

Results from Table 5.7 do not imply any significant performance improvements for occluded objects when including additional input channels to PatchNet. On the contrary, as occlusion level increases to the High category, results favor PatchNet without any additional channels.

5.2 Attention Mechanism Ablation Studies

In this section we will present quantitative results for our attention ablation studies. Performances will be compared as is described in Section 5.1. PatchNet with no additional input channels, trained on objects ranging from 0 to 100 meters, will be included in all tables for comparison. This allows for direct assessments of how replacing the binary mask in PatchNet with variants of our proposed attention mechanism affects model performance.

5.2.1 Attention Heads Ablation

Tables 5.8 and 5.9 show quantitative results from the attention head ablation experiments, described in Section 4.5.1.

	$AP\|_{40}$@ BEV IoU 0.5			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.30	0.08	**0.02**
PatchNet	1.41 ± 0.02	0.50 ± 0.01	0.10 ± 0.00	<u>0.00</u> ± 0.00
AHA 1H	**1.41** ± 0.00	0.52 ± 0.00	<u>0.11</u> ± 0.00	0.00 ± 0.00
AHA 2H	1.40 ± 0.04	**0.52** ± 0.01	0.11 ± 0.01	0.00 ± 0.00
AHA 4H	1.35 ± 0.08	0.52 ± 0.01	**0.11** ± 0.00	0.00 ± 0.00
AHA 8H	<u>1.41</u> ± 0.01	<u>0.52</u> ± 0.01	0.10 ± 0.01	0.00 ± 0.00

Table 5.8: $AP|_{40}$@ BEV IoU 0.5 for AHA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

	$AP\|_{40}$@ RCE, threshold 0.05			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.92	0.73	**0.72**
PatchNet	1.53 ± 0.07	1.22 ± 0.02	0.84 ± 0.01	<u>0.02</u> ± 0.00
AHA 1H	<u>1.57</u> ± 0.02	1.26 ± 0.03	0.86 ± 0.00	0.00 ± 0.00
AHA 2H	**1.59** ± 0.06	<u>1.26</u> ± 0.04	0.84 ± 0.05	0.00 ± 0.00
AHA 4H	1.46 ± 0.11	1.26 ± 0.05	<u>0.86</u> ± 0.01	0.00 ± 0.00
AHA 8H	1.56 ± 0.04	**1.28** ± 0.02	**0.87** ± 0.01	0.00 ± 0.00

Table 5.9: $AP|_{40}$@ RCE, threshold 0.05 for AHA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

The general trend observed in tables 5.8 and 5.9, is that the replacing the binary mask in the PatchNet model with our attention module does impact performance.

For example, with only one attention head, AHA 1H significantly outperforms PatchNet in the Mid and Far categories for both metrics. However, no significant improvements are observed for any of the AHA experiments in the near category.

Additionally, the results do not indicate any clear trend in terms of how the number of attention heads impacts performance, and as such, no clear conclusion can be drawn regarding an "optimal" number of attention heads. While the eight-headed approach looks promising, especially in the RCE setting, one also has to consider that each head contributes with approximately 2 million additional parameters (see Table 4.6). As such, for the following experiments we will employ 4 attention-heads, as a compromise between model potential and the general desire to limit model complexity. However, we want to emphasize our awareness that this choice is not very well grounded from a quantitative perspective.

5.2.2 Attention Dimensionality Ablation

In addition to limiting the number of attention heads, reducing the dimensionality of the embedding, queries, keys and values is also a means of reducing model complexity. Tables 5.12 and 5.13 show numerical results from the attention dimensionality ablation experiments described in Section 4.5.2.

	$AP\|_{40}$@ BEV IoU 0.5			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.30	0.08	**0.02**
PatchNet	1.41 ± 0.02	0.50 ± 0.01	0.10 ± 0.00	<u>0.00</u> ± 0.00
ADA 4H 512×512	1.35 ± 0.08	0.52 ± 0.01	<u>0.11</u> ± 0.00	0.00 ± 0.00
ADA 4H 512×256	**1.42** ± 0.01	<u>0.52</u> ± 0.00	0.11 ± 0.00	0.00 ± 0.00
ADA 4H 512×128	1.41 ± 0.00	0.52 ± 0.00	**0.11** ± 0.00	0.00 ± 0.00
ADA 4H 128×64	<u>1.42</u> ± 0.00	**0.52** ± 0.01	0.11 ± 0.00	0.00 ± 0.00

Table 5.10: $AP|_{40}$@ BEV IoU 0.5 for APA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

	$AP\|_{40}$@ RCE, threshold 0.05			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.92	0.73	**0.72**
PatchNet	1.53 ± 0.07	1.22 ± 0.02	0.84 ± 0.01	0.02 ± 0.00
ADA 4H 512×512	1.46 ± 0.11	1.26 ± 0.05	0.86 ± 0.01	0.00 ± 0.00
ADA 4H 512×256	1.60 ± 0.06	**1.29** ± 0.00	0.86 ± 0.02	0.00 ± 0.00
ADA 4H 512×128	1.59 ± 0.01	1.28 ± 0.02	**0.87** ± 0.01	0.00 ± 0.00
ADA 4H 128×64	**1.64** ± 0.03	1.29 ± 0.01	0.86 ± 0.01	0.00 ± 0.00

Table 5.11: $AP|_{40}$@ RCE, threshold 0.05 for APA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

The results in tables 5.10 and 5.11 show that the performance gains of our proposed attention mechanism are not overly reliant on having high dimensionalities. For example, ADA 4H 512×512 does not significantly outperform ADA 4H 128×64 in any category, even though the latter nearly halves the number of trainable parameters in the model (see Table 4.7).

Noting that the feature vector bottleneck dimensionality for APA 4H 128×64 is 256, we proceed by working with the APA 4H 512×128 set-up, which maintains a dimensionality of 512 for the pooled feature vector, which is equivalent to PatchNet. This model was seen to perform well for both metrics, while still yielding a remarkable reduction in the number of parameters.

5.2.3 Attention Pooling Ablation

In [19], the authors perform an ablation study on the pooling mechanism used for the model illustrated in Figure 3.4, concluding that global max pooling performs better than global average pooling. Tables 5.13 and 5.12 show model performance for the global average- and max pooling, when also including the attention module described in Section 3.5.

	$AP\|_{40}$@ BEV IoU 0.5			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.30	0.08	**0.02**
PatchNet	1.41 ± 0.02	0.50 ± 0.01	0.10 ± 0.00	0.00 ± 0.00
APA MP 4H	1.39 ± 0.03	0.49 ± 0.03	0.10 ± 0.01	0.00 ± 0.00
APA AP 4H	**1.41** ± 0.00	**0.52** ± 0.00	**0.11** ± 0.00	0.00 ± 0.00

Table 5.12: $AP|_{40}$@ BEV IoU 0.5 for APA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

	$AP\|_{40}$@ RCE, threshold 0.05			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.92	0.73	**0.72**
PatchNet	1.53 ± 0.07	<u>1.22</u> ± 0.02	0.84 ± 0.01	<u>0.02</u> ± 0.00
APA MP 4H	<u>1.53</u> ± 0.04	1.22 ± 0.04	<u>0.84</u> ± 0.05	0.00 ± 0.00
APA AP 4H	**1.59** ± 0.01	**1.28** ± 0.02	**0.87** ± 0.01	0.00 ± 0.00

Table 5.13: $AP|_{40}$@ RCE, threshold 0.05 for APA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

The results in tables 5.12 and 5.13 indicate no advantage to using max pooling instead of average pooling. With the inclusion of our proposed attention module, we instead observe that average pooling yield a performance improvements over max pooling. For example, this can be seen seen when comparing APA MP 4H with APA AP 4H in the RCE setting, where the latter is significantly better in the Near category.

5.2.4 Attention Input Ablation

Tables 5.14 and 5.15 show results from incorporating additional input channels from RGB and aleatoric uncertainty, while using the APA 4H 512×128 hyperparameters (tabulated as AIA XYZ here), described in Table 4.9.

	$AP\|_{40}$@ BEV IoU 0.5			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.30	0.08	**0.02**
PatchNet	1.41 ± 0.02	0.50 ± 0.01	0.10 ± 0.00	<u>0.00</u> ± 0.00
AIA XYZ	1.41 ± 0.00	0.52 ± 0.00	0.11 ± 0.00	0.00 ± 0.00
AIA RGB	**1.43** ± 0.02	**0.58** ± 0.02	**0.13** ± 0.01	0.00 ± 0.00
AIA RGB+Unc	<u>1.42</u> ± 0.01	<u>0.53</u> ± 0.01	<u>0.11</u> ± 0.00	0.00 ± 0.00
AIA Unc	1.40 ± 0.04	0.52 ± 0.02	0.11 ± 0.00	0.00 ± 0.00

Table 5.14: $AP|_{40}$@ BEV IoU 0.5 for AIA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

	$AP\|_{40}$@ RCE, threshold 0.05			
Distance	Near	Mid	Far	Very Far
Baseline	1.00	0.92	0.73	**0.72**
PatchNet	1.53 ± 0.07	1.22 ± 0.02	0.84 ± 0.01	0.02 ± 0.00
AIA XYZ	1.59 ± 0.01	1.28 ± 0.02	0.87 ± 0.01	0.00 ± 0.00
AIA RGB	**1.64** ± 0.03	**1.38** ± 0.04	**0.99** ± 0.09	0.00 ± 0.00
AIA Unc	1.60 ± 0.06	1.31 ± 0.01	0.88 ± 0.01	0.00 ± 0.00
AIA RGB+Unc	1.60 ± 0.01	1.32 ± 0.01	0.88 ± 0.01	0.00 ± 0.00

Table 5.15: $AP|_{40}$@ RCE, threshold 0.05 for AIA experiments, split by distance. Tabulated values are all divided by the baseline $AP|_{40}$ score in the Near category, due to confidentiality reasons.

From the data in tables 5.14 and 5.15, it is clear that AIA RGB yields significantly improved performance over PatchNet in all categories except the BEV Near setting, where the error margins overlap. Moreover, AIA RGB also significantly outperforms the other AIA input experiments in the Mid and Far settings, for both metrics.

Similarly to what was seen in the PatchNet input ablation study, the inclusion of the uncertainty input also seems to bring some performance benefit to the attention mechanism. However, in comparison to the AIA RGB-experiment, these differences are much less significant. As such, we once again conclude that RGB-input aids the model most significantly among the tested additional input channels.

Tables 5.16 and 5.17 shows the network performances with respect to varying occlusion.

	$AP\|_{40}$@ BEV IoU 0.5			
Occlusion	No	Low	Med	High
Baseline	1.00	0.37	0.24	0.05
PatchNet	1.90 ± 0.02	0.77 ± 0.02	0.42 ± 0.02	0.08 ± 0.01
AIA XYZ	1.91 ± 0.00	0.79 ± 0.00	0.41 ± 0.03	0.07 ± 0.01
AIA RGB	**2.00** ± 0.01	**0.82** ± 0.03	**0.45** ± 0.01	0.07 ± 0.02
AIA Unc	1.91 ± 0.03	0.79 ± 0.02	0.42 ± 0.02	0.07 ± 0.01
AIA RGB+Unc	1.93 ± 0.01	0.79 ± 0.02	0.42 ± 0.02	**0.08** ± 0.01

Table 5.16: $AP|_{40}$@ BEV IoU 0.5 for AIA experiments, split by occlusion. Tabulated values are all divided by the baseline $AP|_{40}$ score in the No occlusion category, due to confidentiality reasons.

	$AP\|_{40}$@ RCE, threshold 0.05			
Occlusion	No	Low	Med	High
Baseline	1.00	0.54	0.46	0.19
PatchNet	1.44 ± 0.05	0.68 ± 0.03	0.56 ± 0.02	0.23 ± 0.00
AIA XYZ	1.48 ± 0.00	0.73 ± 0.01	0.58 ± 0.01	0.23 ± 0.00
AIA RGB	**1.54** ± 0.03	**0.75** ± 0.03	**0.63** ± 0.03	**0.25** ± 0.02
AIA Unc	1.48 ± 0.03	0.72 ± 0.01	0.60 ± 0.02	0.23 ± 0.00
AIA RGB+Unc	1.49 ± 0.00	0.72 ± 0.01	0.58 ± 0.01	0.23 ± 0.00

Table 5.17: $AP|_{40}$@ RCE, threshold 0.05 for AIA experiments, split by occlusion. Tabulated values are all divided by the baseline $AP|_{40}$ score in the No occlusion category, due to confidentiality reasons.

The most significant increase in performance of AIA RGB in comparison to both PatchNet and AIA XYZ seems to be for non-occluded objects, in both the BEV and RCE setting. While AIA RGB performs better than PatchNet and AIA XYZ on average for the Low and Med occlusion categories, we observe no significant improvements in the BEV setting. However, in the RCE setting, AIA RGB is significantly better than PatchNet in the Med category.

Having observed the performance increase of AIA RGB over other models, an important aspect to address is the fact that the AIA RGB model contains significantly more parameters than the PIA RGB experiment, which also used RGB but had the same number of parameters as PatchNet. As such, it is highly relevant to investigate whether the increase in performance stems from the introduced attention mechanism, or if it is a result of the increased model capacity induced by the additional parameters in combination with the added RGB input. To address this issue, we performed additional experiments with a larger backbone network for PIA RGB, replacing the ResNet-18 with the larger ResNet-34, resulting in approximately matched parameter counts to the attention mechanism. Experimental results are shown in Table 5.18.

		$AP\|_{40}$@ BEV IoU 0.5		
Distance	# params [10^6]	Near	Mid	Far
Baseline	-	1.00	0.30	0.08
PIA RGB - ResNet-18	3.9	1.42 ± 0.00	0.52 ± 0.01	0.12 ± 0.01
PIA RGB - ResNet-34	5.2	1.38 ± 0.02	0.50 ± 0.03	0.10 ± 0.02
AIA RGB - ResNet-18	5.3	**1.43** ± 0.02	**0.58** ± 0.02	**0.13** ± 0.01

Table 5.18: $AP|_{40}$@ BEV IoU 0.5 for different number of parameters. Here, we compare AIA RGB with PIA RGB. To make up for the difference in number of parameters, the PIA RGB experiment uses the larger ResNet-34 backbone network.

From Table 5.18, it is clear that introducing additional parameters in the backbone

network does not match the performance gains from our attention heads. As such, we can attribute the increased performance of AIA RGB to our attention mechanism, rather than the general increase in model capacity.

5.3 Comparative Quantitative Results

The previous sections provided quantitative results for all experiments, and found a very promising set-up by combining our attention mechanism and additional RGB channels (AIA RGB). Here, we want to provide a higher level overview by comparing the performance of our representative hyperparameter choices. Comparisons will be in the $AP|_{40}$ BEV and RCE settings. However, unlike in previous sections, here we consider *overall* performance, corresponding to evaluation on *all* objects in the evaluation set, without any filtering based on distance or occlusion. Furthermore, for ease of comparison, we present the highest observed metric value during experimentation (for each hyperparameter set up, we choose the best model from the three corresponding experiments). These results are summarized in Table 5.19. Note that these results are normalized with respect to the best PatchNet performance, and not the previously used Baseline. Additionally, we provide the mean and standard deviation of the PatchNet runs normalized by the highest PatchNet metric values.

Overall metrics		$AP\|_{40}$ @ BEV IoU > 0.5	$AP\|_{40}$ @ RCE < 0.05
Baseline	Baseline	0.505	0.752
PatchNet	PatchNet (max)	1.000	1.000
	PatchNet ($\mu \pm \sigma$)	0.989 ± 0.020	0.991 ± 0.022
	PIA RGB	1.013	1.029
	PIA Unc	1.002	1.012
	PIA RGB+Unc	<u>1.031</u>	<u>1.080</u>
Ours	AIA XYZ	1.004	1.017
	AIA Unc.	1.013	1.031
	AIA RGB+Unc	1.017	1.064
	AIA RGB	**1.072**	**1.138**

Table 5.19: Here, we present overall performance of the *best* runs for different hyperparameters. For more details and statistics, refer back to the sections describing the respective ablation studies. The respective columns are divided by the absolute AP score for the best trained PatchNet baseline. For clarity, we also include the mean PatchNet baseline performance, with error margins corresponding to one standard deviation.

The most interesting observation from the table is that some of our proposed modifications clearly outperform PatchNet overall. While simply including additional channels, for example in the PIA RGB+Unc model, seemingly raises performance,

the absolute highest performance we obtained was with the AIA RGB setup. As a reminder, this model has four attention heads, with dimensionalities $d_e = 512$, $d_k = 128$, $d_q = 128$ and $d_v = 128$, and the additional RGB-input channels.

It is important to address the significance of the observed best result for AIA RGB, seeing as it is only one value. Considering the mean 0.989 and standard deviation 0.020 of PatchNet in the BEV setting, the p-value of obtaining a comparable result to the 1.072 obtained by AIA RGB is in the order of 4σ above the PatchNet mean, or to put it simply, very unlikely. Similarly, for the RCE setting, AIA RGB convincingly outperforms PatchNet, with a metric value 6σ above the PatchNet mean.

5.4 Qualitative Results

In the introduction of this work, we hypothesized that the binary mask in PatchNet [19] can be improved by means of an attention mechanism. In this section, we illustrate some qualitative results, comprising of examples of attention maps generated from our attention mechanism. We highlight trends in the attention maps, as well as differences to the corresponding binary masks. Note that, due to the qualitative nature of these results, the observations may tend to be subjective.

Figure 5.1 shows examples generated by the best AIA RGB model. The first column shows the RGB-input, the second column shows the binary mask generated from the corresponding depth map, and the subsequent four columns show the generated attention maps. The corresponding depth map is not shown due to confidentiality agreements.

Figure 5.1: Visualization of binary mask and the four attention masks for AIA RGB, for non-occluded objects. Stronger pink colour in a pixel indicates large weight in the mask.

In the above figure, we notice a few trends. First and foremost, we see that for the non-occluded objects, the binary mask successfully selects the pixels corresponding to the object in question. However, in some occurrences, for example the fourth row, the binary mask includes points belonging to the occluding object and discards points belonging to the car in focus. For the patch in the fifth row, this effect is even stronger, where nearly all retained points belong to the occluding objects, and the car of interest is almost completely discarded. When looking at the attention maps, it is evident that the respective heads consistently attend to similar parts of vehicles across different patches. For example, Head 1 generally attends to the bottom of the car, Head 2 to the roof, Head 3 to the sides while Head 4 attends to larger parts of the vehicle body. Note that this behaviour is consistent, even for the occluded objects where the binary mask performed poorly (i.e., the fourth and fifth row).

Notably, the binary mask seems much less sparse than the corresponding attention maps. However, one must keep in mind that in PatchNet [19], global max pooling is applied to the masked feature maps, thus only one pixel from each feature map is selected for the feature vector. In the case of the attention mechanism, average pooling weighted by the attention maps is performed on the transformed feature maps. As such, each pixel in the feature maps contribute to the feature vector, with some pixels contributing more strongly due to being more strongly attended to.

Figure 5.2 shows further examples of binary masks and attention maps, here also for heavily occluded objects.

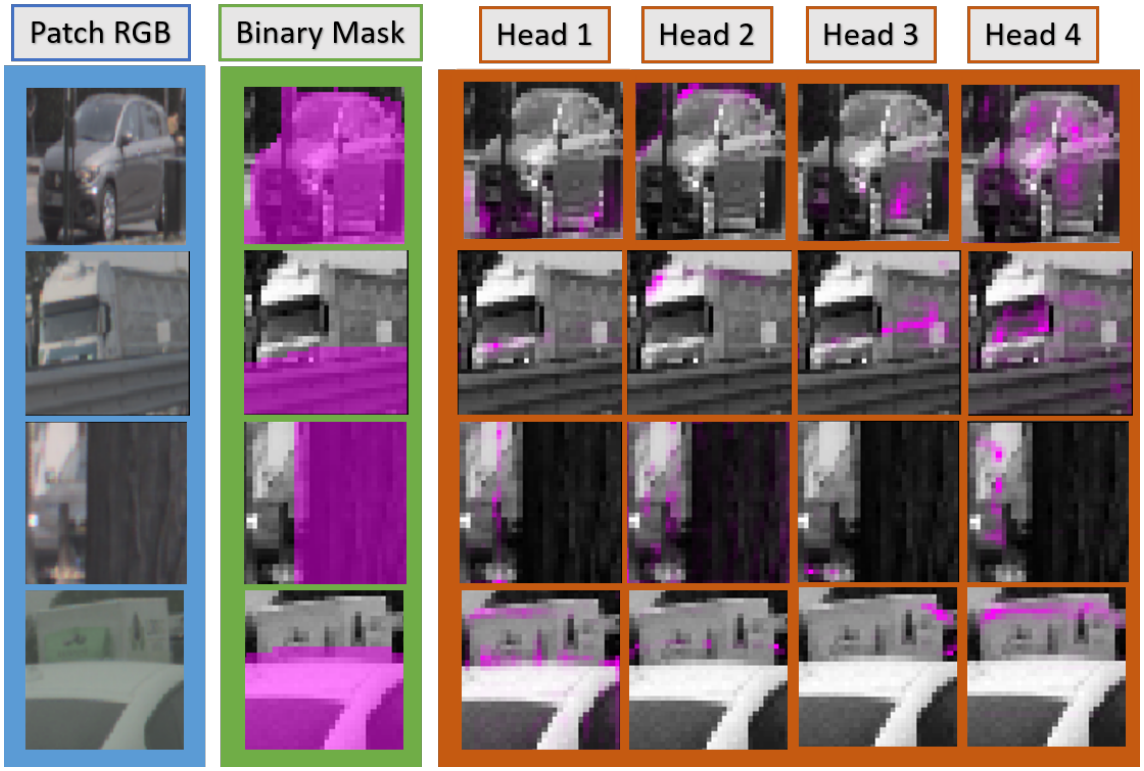


Figure 5.2: Visualization of binary mask and the four attention masks for AIA RGB, for occluded objects. Stronger pink colour in a pixel indicates large weight in the mask.

From the above figure, it is clear that the binary mask primarily selects points belonging to the occluding object as occlusion becomes more drastic. This can be intuitively understood by remembering that the binary mask is calculated by means of a depth threshold, set as the mean depth across the patch plus a constant offset, discarding all point with depth values above this threshold. On the other hand, our attention module allows the model to selectively attend to the pixels which belong to the object of interest, such that it can completely disregard the occluding object. For example, in the second row, our attention maps put very little weight on the occluding barrier. Instead, it successfully attends to the occluded object, with Head 2 maintaining its trend of looking at the roof of the vehicle, despite the difficulty of the example. However, overall, the trends are not as clear for these occluded objects, as they were for non-occluded objects in Figure 5.1.

Figure 5.3 shows a comparison between the number of heads employed in the multi-head attention mechanism. For fair comparison, the models employed for these visualizations are AHA 4H and AHA 1H.

Figure 5.3: Visualization of binary mask and attention masks for AHA experiments. Stronger pink colour in a pixel indicates large weight in the mask. Here, masks from AHA 1H and AHA 4H are shown, to compare how number of attention heads affects mask interpretability.

From the above figure it is clear that availability of multiple attention heads allows each head to attend to more specific features of a patch. For example, the first head in the AHA 4H examples seems to attend to the bottom of the car, whereas the AHA 1H attention maps distribute their weight around the entirety of the car. As such, increasing the number of heads in the attention mechanism seems to help the network to selectively attend to more specific parts of the patch, yielding more tailored features, while the one attention head of AHA 1H seems to have to attend

to all components simultaneously.

Since the addition of the RGB channels brought significant increases in performance for the attention model, we also visualize some qualitative differences between AIA RGB and AIA XYZ in Figure 5.4. The top three rows show the results from models only receiving the XYZ input, and the bottom three rows show the results from models receiving the additional RGB-input.

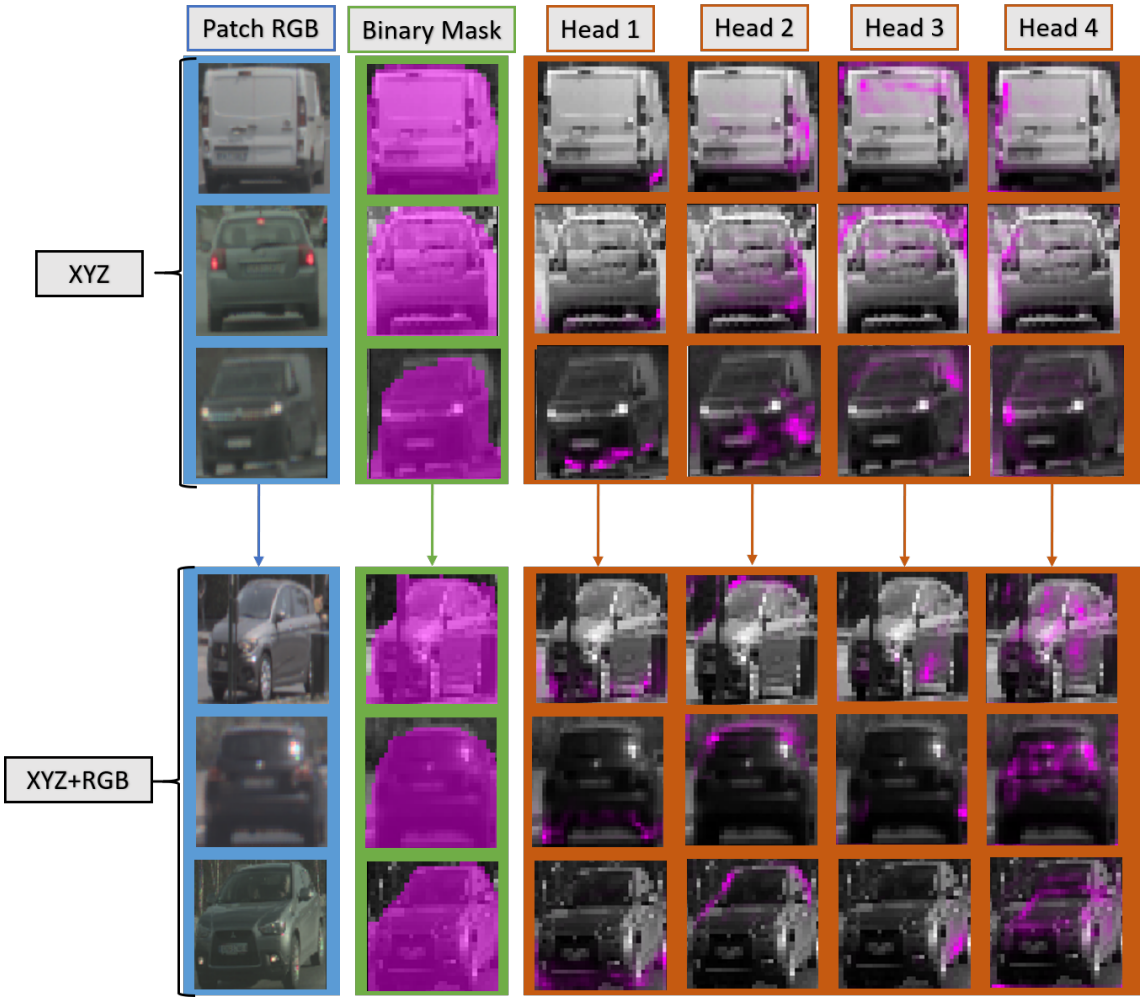


Figure 5.4: Visualization of binary mask and attention masks for AIA experiments. Stronger pink colour in a pixel indicates large weight in the mask. Here, masks from AIA XYZ and AHA RGB are shown, to compare how number of attention heads affects mask interpretability.

In the above figure, it becomes clear that both AIA RGB and AIA XYZ selectively attend to quite distinct regions in the patches. The general trend seems to be that the inclusion of RGB yields slightly more detailed and less blurry attention maps than its XYZ-only counterpart. For example, Head 2 for AIA RGB has very sharp focus. However, some examples from the XYZ-only attention maps are also quite sharp, for example those produced by Head 4. As such, these last few observations should be taken as more speculative.

6

Discussion

In this chapter, we discuss the results from Chapter 4 in relation to our research questions. We also provide comment and discussion around our chosen methods, and possible future work.

6.1 Analysis of Results

Our first research question was aimed at addressing how PatchNet performs with increasing distances. The studies conducted in this work show that detection performance decreases drastically with increased depths, an effect which is seen across all experiments. We have observed that both detection metrics decline significantly as distance is increased. This is expected, as the problem of precisely localizing an object gets more difficult with increased distance. In Table 2.2, we can see that this is true for models in the literature, with detection performance dropping as the difficulty increases (difficulty is largely determined by distance). While this is true for LiDAR based approaches, the decrease in performance is more rapid for models relying on mono-depth. The most immediate explanation for this is that monocular depth estimations, unlike LiDAR, quickly lose accuracy as distance increases, and consequently, the spatial data on which the 3DOD model is reliant is not sufficiently precise for extracting the information needed for an accurate detection. As such, it becomes clear that the performance of mono-depth based approaches to 3DOD are bounded by the quality of the estimator itself. While our proposed modifications do aid the models at the higher distance categories, the greatest potential for detection improvement at higher distances likely lies in improving the actual depth estimator.

In addition to performance dropping with increased distance, we also observed an effect in our distance ablation study where inclusion of training objects above a certain distance threshold lowered the average performance of the model at closer distances. One plausible explanation is that, above this threshold the quality of the depth estimations in combination with the drop in patch resolution renders the input data too noisy, which has a detrimental effect on the total learning of the network. Given that our model performed poorly compared to the Baseline model at these distances, we decided to exclude objects above these distances in our subsequent experiments, favouring performance in the closer distance categories.

Even when excluding the Very Far examples from the training set, one could ex-

pect the model to generalize above the training distances. The reason why this is expected, is due to the standardizing non-trainable transformations applied to each patch, such that it is centered at the origin. Additionally, the model is constructed to regress residual rather than absolute values for e.g., the center estimation. As an example, even if the model has only trained on objects ranging from 0-30 meters, it should not be limited to outputting only center estimates in this range, which would otherwise cause a lack of generalization capability. In our distance ablation experiment, we observed the exact opposite, namely that models do not generalize to distances beyond their training intervals. This could imply that the data *distribution* of the depth estimations change drastically with increased object distance. The intuitive explanation for this is that objects in the depth map become less distinct with increased distance, whereas closer objects are sharper and more detailed. As such, the model may learn to predict the heading and size of a nearby car, by carefully considering how the depth map is shaped in the corresponding region, since the quality of the depth map in these regions is high. Meanwhile, if a car 100+ meters away is detected, the depth-estimation likely lacks such details.

In our regression head ablation, we tested the effects of varying the number of regression heads, which is shown in [19] to increase the performance of PatchNet. In theory, the reason for this could be closely connected to the discussions above, where each regression head would specialize in a separate distance interval, thus alleviating the problems of distance-induced changes in data distribution. However, in our experiments, we did not notice the same notable difference in performance with varying number of regression heads. On average, multiple heads seems to be slightly favored, but the results are not significant and thus remain unclear. Notably, we observe a large increase in stability across experiments when going from one to two regression heads, regardless of metric. One potential explanation for this would be that the backbone network, which is a common denominator regardless of the number of regression heads, is a bottleneck in this process.

The second research question of this thesis was how the use of additional input channels impact the performance of PatchNet. To simplify the discussion, we consider the addition of RGB channels and aleatoric uncertainty estimates separately. We observed only minor improvements on average when only adding the aleatoric uncertainty estimate as an additional input channel. These estimates likely have a strong correlation with the corresponding depth estimations, which is a theory as to why no larger effects were observed.

Our experiments indicate that including RGB channels to the PatchNet backbone network does, however, significantly improve performance. Specifically, our results show that the model was especially aided by the additional channels at longer distances. Tying back to the discussion about mono-depth quality rapidly decreasing with distance, one hypothesis as to why RGB proves helpful is that it may provide complimentary clues about the spatial position of an object. This information, while not explicit, may be extracted by carefully considering optical distortions in a patch. Consider the two cars in Figure 6.1, captured at different distances.

Figure 6.1: Near and faraway car, shown in left and right, respectively.

Due to the large solid angle of the nearby car, the radial distortion in the spherical camera is very strong. As a consequence, the front and hood of the car is perceived as very large relative to the rest of the car. However, for the faraway car, this effect does not occur, since the radial distortions are not as strong for such a small solid angle. This type of information could be used by the model as an alternative to the explicit monocular-depth, especially in cases where the mono-depth quality is poor.

With regards to different levels of occlusion, our results do not show any clear benefit when supplying RGB as additional input. However, for the input ablation experiments, the binary mask was still used. In our subsequent qualitative study of the attention mechanism, we observed that this mask often discards occluded objects from the feature maps. As such, it is expected that the additional input channels do not benefit the network in any significant way with regards to occlusion.

In our final set of research questions, we asked whether the performance of PatchNet could be improved by means of replacing the binary mask with an attention mechanism. From our experiments, we observed that our proposed attention mechanism improves performance for many different hyperparameter set-ups. The greatest improvement was seen for our AIA RGB model, described in Section 5.2.4, which also utilized additional RGB channels, in addition to our attention mechanism. With this model, we increased performance over PatchNet by more than 7% in the overall BEV setting, and more than 13% in the overall RCE setting (see Section 5.3 for more details).

The quantitative performance gains for AIA RGB were especially convincing for non-occluded objects. We also observed clear qualitative results in Figure 5.1, indicating that the model can consistently identify features such as the roof and wheels of a car when they are visible. In contrast, the binary mask global pooling operation does not directly utilize such distinctive features. As such, one possible explanation for the increase in performance is that, in cases where the network can successfully identify the roof, wheels and side of a car, the distinctive nature of these features yields an especially informant set of features in the feature vector.

In addition to the performance observed for non-occluded objects, we also observed interesting results for occluded objects. While these performance gains were not as distinct quantitatively, the quantitative results were perhaps most striking. We

observed in Figure 5.2 that the attention maps are capable of disregarding occluding objects in the foreground, whereas the binary mask from PatchNet would mistakenly filter out large parts of the relevant object. Additionally, the trends seen for each attention head and which features they attended to seemed to transfer to occluded objects somewhat well.

As to why the quantitative performance with regards to occluded objects are not as clear as for non-occluded objects, we have two main theories. Firstly, the quality of the depth map around occlusion boundaries is poor. Due to occlusion boundaries effectively being depth gradient singularities, training a mono-depth estimator to produce sharp edges is extremely challenging [51]. As a consequence, even if the model selectively attends to pixels belonging to the relevant object by utilizing the RGB channels, the depth map may provide very limited information for heavily occluded objects due to its blurriness around the edges of the foreground object. Secondly, the type of features extracted by the attention mechanism may not be very robust for occluded objects. Given that the parts of the image which each attention head attends to seems to be quite consistent, e.g., one head consistently attends to the wheels/bottom of the car, then, when these parts of the objects are occluded, the attention mechanism may produce signals which are either noisy or misguiding. This can be exemplified by the occluded truck on the second row of Figure 5.2, where the wheels of the truck are occluded, yet the attention map still seems to attend to the lower part of the visible portion of the truck, perhaps producing a inaccurate signal regarding the location of the wheels.

One caveat of our proposed attention module is that it increases the number of parameters in the baseline by about 33%. We excluded the possibility our performance improvements as a consequence of the increased number of parameters by introducing a larger backbone-network in place of the attention module which approximately equalized the number of parameters. Since the network with the larger backbone ResNet did not achieve the same performance, we concluded that it is *indeed* our proposed attention mechanism which improves performance. Regardless, the prospects of real-world applications become more distant as we increase the number of parameters in the model. One needs to remember that this work is directed towards the development of autonomous vehicles, where fast inference speeds are crucial in order to build a safe and reliable system. As such, it is not self-evident that the increments in terms of performance on selected metrics presented in this work make up for the consequent loss in inference speed from introducing additional parameters to our baseline model.

6.2 Analysis of Methods

To put it bluntly, choosing the right hyperparameters for complex models such as the ones studied in this work is nearly impossible. With a list of more than 100 hyperparameters, each with multiple options, one would need immense computational resources to even come close to finding an "optimal" combination. Our means of navigating the complex hyperparameter-space was based in a set of ablation stud-

ies. In these studies, we varied a single component, or hyperparameter, at a time, allowing us to asses whether this component has a positive or negative impact on the model as a whole. At their core, ablation studies work well if the variables are separable, and do not affect each other. In practice, this is likely not the case, rendering the ablation methodology an easy target of criticism. However, it was crucial to this work and our ability to arrive at a somewhat experimentally backed model setup.

A more specific weakness of the ablation methodology is that the specific ordering of our ablations has likely had an impact on our results and conclusions. As an example, consider the attention heads- and attention dimensionality ablation studies. The heads study, which was performed first, yielded quite inconclusive results, resulting in a not-so-rigorous parameter choice. On the other hand, we saw more significant trends in the dimensionality ablation. Since computational resources limited us from going back to the heads ablation after choosing a dimensionality setup, there is absolutely nothing guaranteeing that we had come to the same conclusion regarding optimal model parameters if we conducted the experiments in opposite order.

A major weakness of our results are that they are based on a small sample size in terms of the number of runs per experiment, which was necessitated by the computational resources needed for each training run. As a result, many of our results are hampered by noise and non-significance. As such, a more streamlined method may have allowed us to produce more training runs for the most promising experiments at the cost of less interesting experiments. However, naturally, this is difficult to foresee a priori. We also note that a trend in literature seems to be to report only the best observed performance, which is appropriate given that results are often reported on standardized benchmarking data sets. In our case, we collected a new data set, on which we evaluated our models. As such, our results should only be taken as valid for this setting, and further testing on proper benchmarking data should be performed in order to get a clearer view of how the proposed model performs in different settings, as compared to other models in the literature.

Our method for producing our data set also comes with some weaknesses. As a reminder to the reader, we produce our set of patches by running inference on our image data set using the pre-trained 2DOD network, selecting those detections which achieve an IoU > 0.5, and cropping those detections from the input images/mono-depth outputs. As such, for each patch, we obtain a corresponding ground truth target, forming a set of patch-ground truth pairs. For our subsequent validation metrics, we calculate the average precision based on this set of patches. A more rigorous method of validating, which is widely used for two-stage detectors on the KITTI benchmark [19, 7], would be to run inference with the entire pipeline, including 2D detection stage, on each full image, and produce a set of detections for each image. TPs, FPs, etc., are then identified by comparing the set of detections with the set of ground truths in the image. This ensures that the FPs of the 2DOD network are accounted for when calculating subsequent metrics, which is not the case in our validation methodology. However, since we treat the 2DOD network as-is during this work, all models we validate use the exact same set of 2D detections,

and produce exactly one 3D bounding box for each such detection. As such, the reader should keep in mind that with this set-up, we exclusively evaluate how modifications to the 3D regression branch (see Section 3.4) affect the overall performance of the pipeline.

Our results are only generated based on the vehicle object class. A similar delimitation is common in the literature, with most comparisons on the KITTI data set highlighting performance on the Car category [7, 19, 18]. Additionally, it also allowed for a simpler metric and ease of computation. However, the results should be read with this in mind, and further evaluation on more object classes should be performed in order to ascertain real-world feasibility.

As mentioned previously, results presented in this work are on our validation set, rather than a dedicated test set. This was necessitated by the fact that there is no test set for the Baseline network. Clearly, not having a dedicated test set is a weakness of this work. The primary drawback in our case is that the use of early stopping may introduce a bias towards the validation set. To elaborate, the models are thus likely to perform better on the evaluation set than they would on a completely unseen test set. The absence of a test set thus forces us to only draw comparative conclusions, i.e., which of our models performs best on the validation set. If a test set representative of the 3DOD problem had been used, this work could have been extended to assess real-world capabilities.

Other than early stopping, the only hyperparameters which were studied were those varied in our ablation studies. As such, our hyperparameter search was very limited, considering that most of the parameters in our model were never altered, but just kept as was proposed in the PatchNet paper [19]. The absence of a test set forces us to only draw comparative conclusions, i.e., which of our models performs best on the validation set. If a test set representative of the 3DOD problem had been used, this work could have been extended to assess real-world capabilities.

When evaluating 3DOD models, the choice of metric is vital for presenting fair results. Since the 3D bounding boxes have position, size, rotation, etc., finding a metric which justly accounts for the importance of the respective parameters is very difficult. The $AP|_{40}$ @ 3D IoU 0.7, commonly used in the literature, takes into account most of the 3D box parameters. However, the metric is unforgiving to (relatively) small positional errors at longer ranges and does not take into account the 180° yaw modality of the bounding box, whereas one could argue that the acceptable positional error should grow with distance and that the 180° yaw modality is vital due to the importance of knowing a vehicle is heading towards- or away from you. In this work, we use metrics based RCE and BEV IoU (section reference). As such, our metrics reflect not only the quality of center estimates, but also the quality of the size, and orientation of the box. However, we are still vulnerable to the aforementioned 180° yaw modality. Figure 6.2 illustrates the distribution of absolute yaw errors for one of our PatchNet experiments.

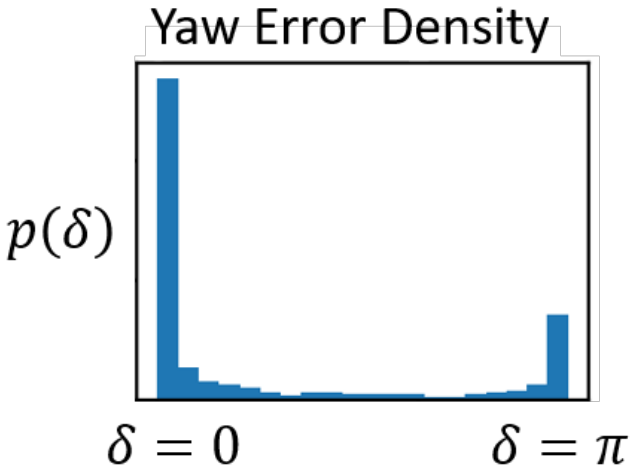


Figure 6.2: Histogram over distribution of yaw angle errors ($\delta = \left|\alpha^{\text{yaw}}_{\text{pred}} - \alpha^{\text{yaw}}_{\text{GT}}\right|$) for PatchNet. The key thing to note here is that there are two distinct peaks, one at $\delta = 0$, and one at $\delta = \pi$. This means that quite often, the model mistakes the heading of the car by 180°.

Clearly, flipped predictions do occur relatively frequently for PatchNet, as seen by the peak at $\delta = \pi$ and as such, we highlight a flaw with the metrics used in this work, since this result cannot be identified from the presented quantitative results. A more detailed investigation could be done by considering an metric for the angle regression, such as the average orientation similarity [1]. Closely tied to this, one could speculate that determining the 180° modality for an object is much more difficult from a depth map, as compared to an RGB image. While it would have been interesting to study how the inclusion of RGB channels affected the distribution of errors, no studies to this matter were conducted in this work.

Another weakness in our methods set due to constraints regarding computational resources, is that we did not have time to revisit the distance ablation study with our final model. From our distance ablation study, we concluded that it was reasonable to exclude training objects from the Very Far category, due to the detrimental effects their inclusion had on the performances of the network at shorter distances. This lead to a general weakness in the following experiments, as any eventual performance increases in this category brought by the additional input data and/or the attention mechanism remain unknown. In hindsight, it might have been interesting to include the Very Far category during training for the AIA RGB experiments for example, since we saw impressive performance gains over the baseline in the other higher-distance categories.

While literature commonly uses more intricate learning rate schedulers when training models with attention [37, 52], we observed very little impact on performance from our choice of scheduler. Also, we observed improved performance when somewhat reducing the dimensionalities of the attention mechanism in the dimensionality ablation. This immediately presents an especially good case for further regularizing the

model during training. While we tried introducing some additional regularization techniques, we did not have time to thoroughly evaluate different hyperparameter options for regularizing the attention mechanism. Considering that hyperparameters in the PatchNet architecture have been thoroughly investigated in previous work [19], the regularization and training-related hyperparameters for the attention mechanism are most likely to benefit from further investigations.

We also observed that the normalization/scaling method used for the keys and queries had a crucial impact on performance. we observed that normalization not only affects the qualitative appearance of the resulting attention maps, but also the quantitative performance and training of the mechanism. In our final mechanism, we employ a normalization of the query vector; i.e., we divide each query vector by its Euclidean norm. The main reason for doing this is to reduce the effect of a pixel achieving a high attention activation simply due to the norm of the corresponding query/key vector having a large norm, and rather allow the alignment of the vectors to have a larger effect on the attention activation. Given this reasoning, we also experimented with normalizing the key tensors pixel-wise, so that the alignment of query and key would be given an even more significant role. However, we observed that the resulting attention activations did not seem to exhibit any of the differentiation with regards to attended positions that we observed in the final mechanism, and no significant performance benefit. On the opposite end of the spectrum, we also experimented with leaving the key/query tensors as is, i.e., not performing any normalization at all. This resulted in very unstable training, which we attribute to the saturation of the softmax function, as the resulting attention activations were very sparse, often with a single pixel receiving the majority of activations. This was somewhat mitigated by scaling the dot-products with a fixed value, however, this solution was generally unstable and the choice of the scaling parameter also added an additional hyperparameter to be optimized. As such, normalizing only the query vector was deemed the best compromise, and resulted in the best performance.

6.3 Future Work

In this work, we investigated how an attention mechanism could be used to substitute for the binary mask used in PatchNet [19]. Our most promising experimental outcome was obtained by using multiple attention heads, and complimenting the spatial data in each patch with the RGB image channels. Using this set-up, we saw qualitatively in Figure 5.1 that the attention module regularly attended to select features of objects, such as the roof or wheels of a car. However, the results still exhibited blurriness and some inconsistencies across examples. Due to the highly non-linear nature of extracting fine-grained masks, one potential point for future work is to further develop the representation capacity of the attention mechanism. One potential approach would be to use a set of sequential multi-head attention mechanisms, yielding an architecture more reminiscent of the transformers architecture [37].

When it comes to complex models such as the one in this work, interpretability

is an asset. While the attention maps gave quite interpretable results, we sometimes observed slight redundancies between the maps. In order to enhance model interpretability even further, future work could delve into regularizing the attention heads, such that some dissimilarity between the different heads is maintained. As such, the trends observed in the attention maps (e.g., focus on roof, wheels, etc.) might become even more clear. Having more distinct features may also help the model itself become more robust to examples where common characteristics are difficult to identify.

A general trend observed across different ablation studies was the performance boost gained from supplementing the depth-maps with RGB input. However, the method by which this was done in this thesis was very rudimentary, and as such, we see potential future work in incorporating the RGB data using more sophisticated methods. The attention mechanism introduced in this thesis also showed promise in incorporating RGB-data, and enhancing this effect further could be another interesting path forward. One such approach would be to train the attention mechanism separately using the RGB-channels, which could perhaps decouple the training of the backbone from the attention mechanism, thereby increasing the ease of training.

In this work, we delimited ourselves to treating the 2DOD, and mono-depth networks as constant. Recent developments in the literature propose end-to-end solutions for certain Pseudo-LiDAR pipelines [53], allowing 3D box gradients to pass through the entire model pipeline. One clear point for future work, which was not covered in the PatchNet paper either [19], would be to train the proposed architecture end-to-end. This might help the mono-depth network produce depth maps geared more towards the specific use case, perhaps yielding sharper edges around vehicles due to receiving gradients from the 3DOD loss.

7

Conclusion

The primary contributions of this work are two-fold. Firstly, we investigated how PatchNet works in long-distance settings which are much greater than what is present in the KITTI data set (around which the model was originally built). Secondly, we achieve improved performance over PatchNet on the Zenseact data set, by replacing the binary mask in the model with an attention mechanism.

Our first research question was:

How does the mono-depth based 3DOD pipeline PatchNet perform in settings of increasing distance?

From our experiments, we observed that the performance PatchNet is drastically affected by increased distance. At short ranges of 0 to 30 meters, the performance of PatchNet far surpasses the baseline network, where depth estimations are highly accurate. However, as distance increases this performance advantage is seen to diminish, and for objects positioned further than 100 meters from the camera, PatchNet performed worse than the baseline. We attribute this to the increase in error exhibited by mono-depth estimations as distance is increased, yielding poor detection capacity at long ranges. Furthermore, we observed that inclusion of training examples from above 100 meters seemed to degrade the performance of PatchNet in lower categories, which we also attribute to the poor quality of the mono-depth estimations at this range.

The second research question was:

Can additional input channels, such as RGB channels, improve the performance of PatchNet?

RGB and uncertainty estimates were observed to aid the model, especially in the longer distance settings of 30 to 100 meters. In particular, RGB data seemed to provide the most performance benefits, which we attribute to the possibility that this additional data alleviates some of the problems encountered as a result of the diminished mono-depth quality.

Finally, the third research question was:

Can the 3DOD performance of PatchNet [19] be improved, by means of replacing

the binary mask with a soft-attention mechanism? What are the strengths and weaknesses of this mechanism?

On the Zenseact data set, our proposed attention mechanism shows significant performance improvements over PatchNet. We found that the addition of RGB channels gave the best performance. A clear strength of the mechanism, quantitatively seen in our experiments, is its improved overall detection capacity over PatchNet across different distance and occlusion settings. We also observed promising qualitative results, where the attention mechanism consistently selectively focused on specific parts of vehicles, such as the roof or wheels. The primary weaknesses of the proposed mechanism is that it significantly increases the number of parameters in the model, rendering prospects of application to real-time use in autonomous vehicles more distant.

Bibliography

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] W. H. Organization *et al.*, "Global status report on road safety 2018: Summary," World Health Organization, Tech. Rep., 2018.

[4] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Tech. Rep., 2015.

[5] S. S. Shadrin and A. A. Ivanova, "Analytical review of standard sae j3016 «taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles» with latest updates," *Avtomobil'. Doroga. Infrastruktura.*, no. 3 (21), p. 10, 2019.

[6] S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot, "Computer architectures for autonomous driving," *Computer*, vol. 50, no. 8, pp. 18–25, 2017.

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

[8] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–8.

[9] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, 2020.

[10] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 270–279.

[11] I. Alhashim and P. Wonka, "High quality monocular depth estimation via transfer learning," *arXiv preprint arXiv:1812.11941*, 2018.

[12] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep ordinal regression network for monocular depth estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2002–2011.

[13] J. H. Lee, M.-K. Han, D. W. Ko, and I. H. Suh, "From big to small: Multi-scale local planar guidance for monocular depth estimation," *arXiv preprint arXiv:1907.10326*, 2019.

[14] X. Chen *et al.*, "3d object proposals for accurate object class detection," in *Advances in Neural Information Processing Systems*, 2015, pp. 424–432.

[15] X. Chen, K. Kundu, Y. Zhu, H. Ma, S. Fidler, and R. Urtasun, "3D object proposals using stereo imagery for accurate object class detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 5, pp. 1259–1272, 2017.

[16] C. C. Pham and J. W. Jeon, "Robust object proposals re-ranking for object detection in autonomous driving using convolutional neural networks," *Signal Processing: Image Communication*, vol. 53, pp. 110–122, 2017.

[17] B. Xu and Z. Chen, "Multi-level fusion based 3d object detection from monocular images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2345–2353.

[18] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8445–8453.

[19] X. Ma, S. Liu, Z. Xia, H. Zhang, X. Zeng, and W. Ouyang, "Rethinking pseudo-lidar representation," in *Proceedings of the European Conference on Computer Vision*, 2020, pp. 311–327.

[20] A. Simonelli, S. R. Bulò, L. Porzi, P. Kontschieder, and E. Ricci, "Demystifying pseudo-lidar for monocular 3d object detection," *arXiv preprint arXiv:2012.05796*, 2020.

[21] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[22] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[24] Y. Hirose, K. Yamashita, and S. Hijiya, "Back-propagation algorithm which varies the number of hidden units," *Neural Networks*, vol. 4, no. 1, pp. 61–66, 1991.

[25] B. Mehlig, "Artificial neural networks," *arXiv preprint arXiv:1901.05639*, 2019.

[26] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, pp. 2121–2159, 2011.

[27] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2015.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[29] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, 2009, pp. 248–255.

[30] T. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[31] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[32] ——, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[33] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[34] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" *arXiv preprint arXiv:1805.11604*, 2018.

[35] F. Arnez, H. Espinoza, A. Radermacher, and F. Terrier, "A comparison of uncertainty estimation approaches in deep learning components for autonomous vehicle applications," *arXiv preprint arXiv:2006.15172*, 2020.

[36] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[37] A. Vaswani *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5998–6008, 2017.

[38] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for*

Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2019, pp. 4171–4186.

[39] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[40] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision*, 2020, pp. 213–229.

[41] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *International conference on machine learning*, 2019, pp. 7354–7363.

[42] J. Kannala and S. S. Brandt, "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 8, pp. 1335–1340, 2006.

[43] A. Bhoi, "Monocular depth estimation: A survey," *arXiv preprint arXiv:1901.09402*, 2019.

[44] Y. Kuznietsov, J. Stuckler, and B. Leibe, "Semi-supervised deep learning for monocular depth map prediction," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6647–6655.

[45] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

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

[47] A. Simonelli, S. R. Bulo, L. Porzi, M. López-Antequera, and P. Kontschieder, "Disentangling monocular 3d object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1991–1999.

[48] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, vol. 28, 2015, pp. 91–99.

[49] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3D bounding box estimation using deep learning and geometry," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7074–7082.

[50] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[51] M. Ramamonjisoa, Y. Du, and V. Lepetit, "Predicting sharp and accurate occlusion boundaries in monocular depth estimation using displacement fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 648–14 657.

[52] P. Karpov, G. Godin, and I. V. Tetko, "A transformer model for retrosynthesis," in *International Conference on Artificial Neural Networks*, 2019, pp. 817–830.

[53] R. Qian *et al.*, "End-to-end pseudo-lidar for image-based 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5881–5890.

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

CHALMERS
UNIVERSITY OF TECHNOLOGY