uOttawa

# 3D Object Detection for Advanced Driver Assistance Systems

by

Selameab Demilew

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements
for the M.A.Sc. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

**Abstract**

Robust and timely perception of the environment is an essential requirement of all autonomous and semi-autonomous systems. This necessity has been the main factor behind the rapid growth and adoption of LiDAR sensors within the ADAS sensor suite. In this thesis, we develop a fast and accurate 3D object detector that converts raw point clouds collected by LiDARs into sparse occupancy cuboids to detect cars and other road users using deep convolutional neural networks. The proposed pipeline reduces the runtime of PointPillars by 43% and performs on par with other state-of-the-art models. We do not gain improvements in speed by compromising the network's complexity and learning capacity but rather through the use of an efficient input encoding procedure. In addition to rigorous profiling on three different platforms, we conduct a comprehensive error analysis and recognize principal sources of error among the predicted attributes.

Even though point clouds adequately capture the 3D structure of the physical world, they lack the rich texture information present in color images. In light of this, we explore the possibility of fusing the two modalities with the intent of improving detection accuracy. We present a late fusion strategy that merges the classification head of our LiDAR-based object detector with semantic segmentation maps inferred from images. Extensive experiments on the KITTI 3D object detection benchmark demonstrate the validity of the proposed fusion scheme.

## Declaration

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of University of Ottawa's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

**Acknowledgements**

First and above all, I praise God, the Almighty for all the blessings He has sent my way. As I close this challenging yet fulfilling chapter of my life, I would like to thank the many people who have supported me on my journey.

I'll start by expressing my sincerest gratitude to my supervisor Professor Robert Laganière for his insights, suggestions and mentorship. I would also like to thank all members of VIVA Lab; particularly, Dr. Hamed H. Aghdam has been an invaluable resource from the early days of brainstorming till the final days of writing papers.

I have been fortunate to have a productive and engaging collaboration with Synopsys. I take this opportunity to acknowledge the support I received from Chuck P, Dr. Pierre P, Tom A, Tom M and many more.

I am deeply grateful for my wonderful family. I couldn't have gotten this far without your prayers, wise counsel and boundless love. Ayneye, Noah and Tsegenu, I have no words to express your kindness and hospitality.

Last but not least, a big thank you goes to my friends on both sides of the Atlantic.

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| **ADAS** | Advanced Driver Assistance Systems |
| **AP** | Average Precision |
| **BEV** | Bird's Eye View |
| **BN** | Batch Normalization |
| **CNN** | Convolutional Neural Network |
| **CPU** | Central Processing Unit |
| **DSP** | Digital Signal Processor |
| **EV** | Embedded Vision |
| **FN** | False Negative |
| **FP** | False Positive |
| **FPGA** | Field Programmable Gate Array |
| **FPM** | Fast Performance Model |
| **FPN** | Feature Pyramid Networks |
| **FPS** | Frames Per Second |
| **GAN** | Generative Adversarial Network |
| **GPU** | Graphics Processing Unit |
| **HAPS** | High-Performance ASIC Prototyping System |
| **HOG** | Histograms of Gradients |
| **ILSVRC** | ImageNet Large Scale Visual Recognition Challenge |
| **IMU** | Inertial Measurement Unit |
| **IoU** | Intersection over Union |

| | |
|---|---|
| **LiDAR** | Light Detection and Ranging |
| **MAC** | Multiply–Accumulate Operation |
| **MLP** | Multilayer Perceptron |
| **NMS** | Non-Maximum Suppression |
| **PR** | Precision-Recall |
| **RAM** | Random Access Memory |
| **RANSAC** | Random Sample Consensus |
| **ReLU** | Rectified Linear Unit |
| **RISC** | Reduced Instruction Set Computer |
| **RPN** | Region Proposal Network |
| **RV** | Range View |
| **SAE** | Society of Automotive Engineers |
| **SIFT** | Scale Invariant Feature Transforms |
| **SVM** | Support Vector Machines |
| **TOPs** | Trillion Operations Per Second |
| **TP** | True Positive |

# Chapter 1

# Introduction

Transportation is one of the fundamental pillars of modern society. There are about 35.7 million motorized vehicles in Canada [11] and more than a billion worldwide [87]. Despite their immense socioeconomic importance, cars claim more than 1.35 million lives every year [100]. According to a study [4] conducted by the National Highway and Traffic Safety Administration, 94% of all road accidents are linked to human errors such as distraction, incorrect assumptions, impaired driving and lack of sleep. These accidents can be minimized and possibly eliminated in the long run by offloading the driving task to an automated intelligent system. After all, computers can assess and react to a situation much faster and, unlike us, do not experience fatigue or emotions.

Modern vehicles deploy both active and passive systems to mitigate road accidents. Passive systems such as airbags and seat belts engage during a collision to protect the driver and onboard passengers. Unlike passive systems, active systems proactively monitor their environment and act accordingly to prevent an accident or reduce the severity of an inevitable crash. Common active safety mechanisms such as lane departure warning, adaptive cruise control and forward collision warning all fall under the broad term of advanced driver assistance systems (ADAS).

ADAS are critical milestones on the long quest towards fully autonomous vehicles. The Society of Automotive Engineers (SAE) introduced a 6 level scale, ranging from Level 0 or no automation to Level 5, where the vehicle can safely and efficiently maneuver itself

to a user specified destination without human intervention and minimal prior knowledge of the route. Remaining points along the spectrum are shown in Figure 1.1. In addition to the benefits mentioned above, autonomous vehicles are expected to provide mobility to underserved members of our community, significantly shorten commute time and reduce greenhouse gas emissions.

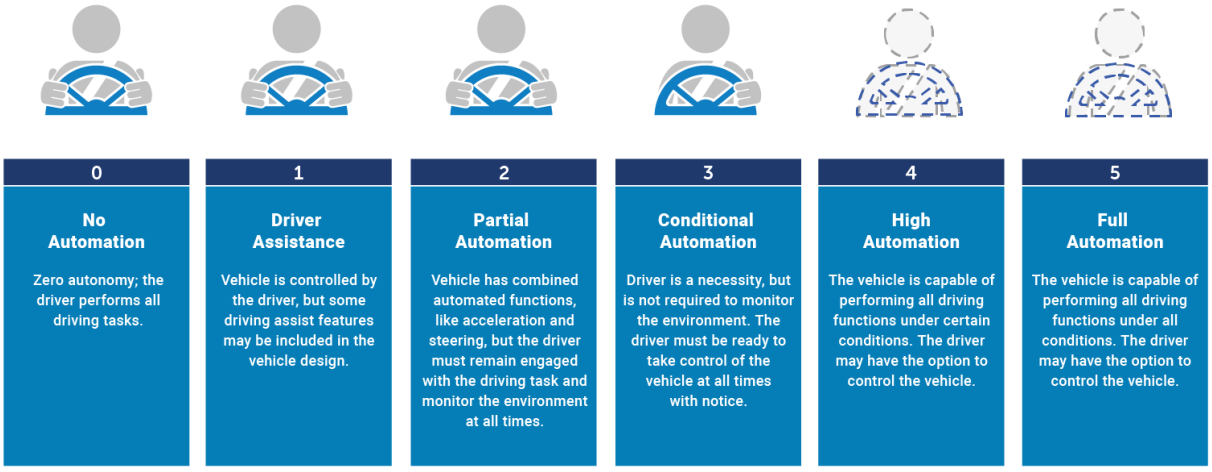



Figure 1.1: Levels of autonomy as defined by the Society of Automotive Engineers

Machine learning, particularly deep learning, has revolutionized the field of computer vision. Ever since the famous work by Krizhevsky *et al.* [46] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [79] by a wide margin, researchers have turned their attention to deep neural networks. Significant advances have been made in various machine vision tasks such as image classification, object and keypoint detection, instance and semantic segmentation, and more recently in panoptic segmentation. Of these, 2D object detection has been a hot area of study for quite some time. CNN based detectors such as YOLO [75, 76] and SSD [61] exhibit high performance on numerous image based detection benchmarks. Nonetheless, mere 2D bounding boxes are insufficient to safely operate a vehicle in a dynamic environment. They lack critical information such as depth and orientation for the system to make accurate time-critical decisions.

## 1.1 Problem Statement

Accurate perception of dynamic objects is a crucial component of all autonomous vehicle pipelines. Any failure to accurately capture this information will often have catastrophic outcomes [33]. This forces the new generation of cars to bundle a plethora of sensors and occasionally retrofit older models. Each sensor is tailored to measure specific attributes of the environment. For instance, cameras capture the visible spectrum of light reflecting off surfaces while light detection and ranging (LiDARs) emit laser beams to estimate depth and create an accurate 3D point cloud. The massive amounts of data produced by these devices needs to be processed in real time. This requirement poses a challenge for mid-range and budget vehicles that employ low-end embedded systems with limited computational power.

The primary goal of this thesis is to build a deep 3D object detector for classifying and localizing road users from point clouds produced by LiDAR sensors. In addition to being robust, the resulting model should be efficient enough to run on compute-constrained embedded devices.

Although LiDARs and cameras can be used independently to detect objects, we hypothesize that combining them will yield better results. To this end, we consider the problem of fusing the information coming from both sensors in order to improve the accuracy of detections.

## 1.2 Challenges

Detecting 3D bounding boxes from LiDAR and images poses several challenges.

- **Limited Computation**: A significant majority of cars are rigged with low-end embedded systems so as to make the technology accessible to a wide range of users. This severely restricts the amount and speed at which incoming data is processed.
- **Irregular Sampling**: As opposed to images, points in a raw LiDAR scan are not sampled at regular intervals. This impedes the use of standard convolutional neural

networks. Moreover, point clouds are sparse and affected by inter-object occlusion as well as self occlusion — where the sensor-facing side obstructs the other side.

- **Multimodal Augmentation**: In order to maintain a coherent and sensible training data, augmentations should be applied to all modalities consistently. Any transformation in the point cloud should be accompanied by an identical transformation in the image.

## 1.3 Contribution

In this thesis, we propose a fast and accurate deep 3D detection network that utilizes 3D occupancy cuboids to encode point clouds. At the time of writing, our model was the fastest deep 3D detection network reported on the ubiquitous KITTI dataset while performing on par with other compute intense detectors. More importantly, we do not improve the execution time by sacrificing the expressive power of the neural network. Instead, we utilize an input encoding that is multiple times faster to compute and a few times faster to transfer to the neural network accelerator. Our main gain in execution time is primarily due to these two factors and not by designing a smaller network. Moreover, we outline techniques to properly train an accurate 3D detection network using this input representation, conduct latency and bandwidth analysis on several hardware and carefully identify central sources of error in the model. Finally, we present a late fusion scheme that enhances the performance of our 3D object detector by fusing its classification head with an image segmentation mask. We extend the proposed fusion strategy to other state-of-the-art 3D object detectors and corroborate the gains in accuracy.

Our main contributions can be summarized as follows:

- A thorough review and comparison of recent literature in 3D object detection
- Compute and bandwidth efficient point cloud encoding and real-time 3D object detector
- Rigorous profiling and optimization on a dedicated embedded vision processor

- An extensive error analysis that identified key sources of error among regressed values
- A late fusion framework for enhancing point cloud detections using image segmentation masks

## 1.4 Publications

The papers listed below were published in collaboration with Hamed H. Aghadam and Robert Laganière as part of the research conducted during this thesis.

- FA3D: Fast and Accurate 3D Object Detection [19]
- RAD: Realtime and Accurate 3D Object Detection on Embedded Systems [5]
- Refine 3D Bounding Boxes through Late Vector Fusion for ADAS Applications [20]

The source code is available at: https://github.com/Selameab/FA3D. Detections from our model on raw KITTI scenes can be found at: https://www.youtube.com/watch?v=LvGY3zKhBEI

## 1.5 Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2** gives a high level overview of the autonomous vehicle software stack and commonly used sensors. Furthermore, it introduces basic principles in machine learning, convolutional neural networks and object detection.
- **Chapter 3** starts by summarizing prior work done on 2D object detectors and follows with an in depth review of the 3D object detection literature.
- **Chapter 4** describes the point cloud data structure, dataset used for our experiments and standard object detection evaluation metrics.

- **Chapter 5** explains the proposed deep 3D object detector (FA3D) in detail. It outlines our input representation, architecture, training procedure, detection accuracy, timing reports and error analysis.

- **Chapter 6** builds upon findings from Chapter 5 and presents a late fusion scheme that uses segmentation maps from images to complement FA3D.

- **Chapter 7** summarizes the thesis and provides possible directions for future work.

# Chapter 2

# Theoretical Background

This chapter presents fundamental concepts that were crucial in developing and analysing the methodologies in this thesis. We start with a high-level overview of the autonomous vehicle software stack followed by a comparison of the ADAS sensor suite. Next, we explain core principles in deep learning, convolutional neural networks and object detection. For a more comprehensive understanding of the subject, we recommend the following resources:

- Self-Driving Cars Specialization [99]
- Deep Learning [31]
- Neural Networks and Deep Learning [67]
- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow [27]

## 2.1 Autonomous Vehicle Software Stack

The autonomous vehicle software stack, shown in Figure 2.1, can be decomposed into four submodules: *Perception*, *Motion Planner*, *Controller* and *Supervisor*. The perception module condenses raw data collected from sensors into a semantic form that is readily processed by subsequent modules in the pipeline. For example, an RGB image is summarized as a list of bounding boxes indicating the location of cars, pedestrians and other relevant

objects. The motion planner searches for the best possible route to take based on constraints set by the user, mechanical constraints set by inherent limitations of the hardware and, most importantly, constraints set by the perception module. The generated trajectory is then executed by the controller. On top of all of these, the supervisor is responsible for coordinating the previously listed modules and alerting the driver in the event of a failure. The detection systems developed in this thesis are part of the perception module.

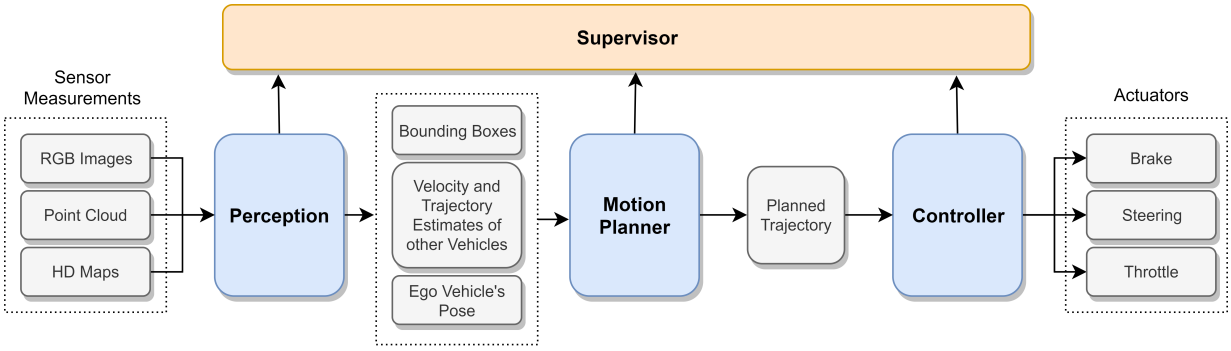



Figure 2.1: Autonomous Vehicle Software Stack

## 2.2 Sensors

Human beings primarily depend on their visual system to drive a car. On the other hand, self driving cars employ a wide range of exterioreceptive sensors to perceive their environment and navigate safely. Cameras, LiDARs, Radars and inertial measurement units (IMUs) are prevalent examples.

The camera is the most widely adopted sensor due to its affordable price. Data captured by a camera (*i.e.* a color image) contains rich texture information. However, it is susceptible to bad weather and changes in lighting conditions. The other drawback of RGB images is the distortion of true physical dimensions. Objects in close proximity to the sensor appear bigger. For example, in Figure 2.2, the cyclist on the right appears to be substantially larger than the both cars.

A LiDAR, which stands for Light Detection and Ranging, is a device that captures a scene in 3D using the time of flight principle. It emits a continuous pulse of light and

Figure 2.2: Distortion of real physical dimensions as a result of projection

estimates distance by measuring the time it takes for the pulse to reach another surface and return to the sensor. The output of a LiDAR is called a *point cloud* and is stored as a set of points. The main benefit of using a LiDAR is that it preserves the physical dimensions of the objects, particularly in bird's eye view. At the moment, LiDARs are expensive. Nonetheless, with the emergence of solid-state LiDARs and economies of scale, the price is likely to decline. The recent integration of LiDARs into tablets for augmented reality [40] is a good indication of the foreseeable future. Another drawback of LiDARs is the variation in sampling density. In Figure 2.3, the LiDAR managed to capture just 10 points corresponding to the car making it nearly impossible to detect. On the other hand, the car is clearly visible in the image.

## 2.3 Computer Vision

Even though it is effortless for humans to visualize, describe and navigate almost any environment, computer vision stands amongst the unsolved challenges of computer science. Traditionally, machine vision was limited to images but now incorporates other modalities such as point clouds. Well-formulated tasks within the field include object detection, classification, segmentation, tracking, pose estimation, image restoration, etc.

Image classification, perhaps the simplest of them all, is the task of determining the class of the dominant object present in an image. Usually, the image will contain a single item positioned around the center. Object detection is another classic machine vision

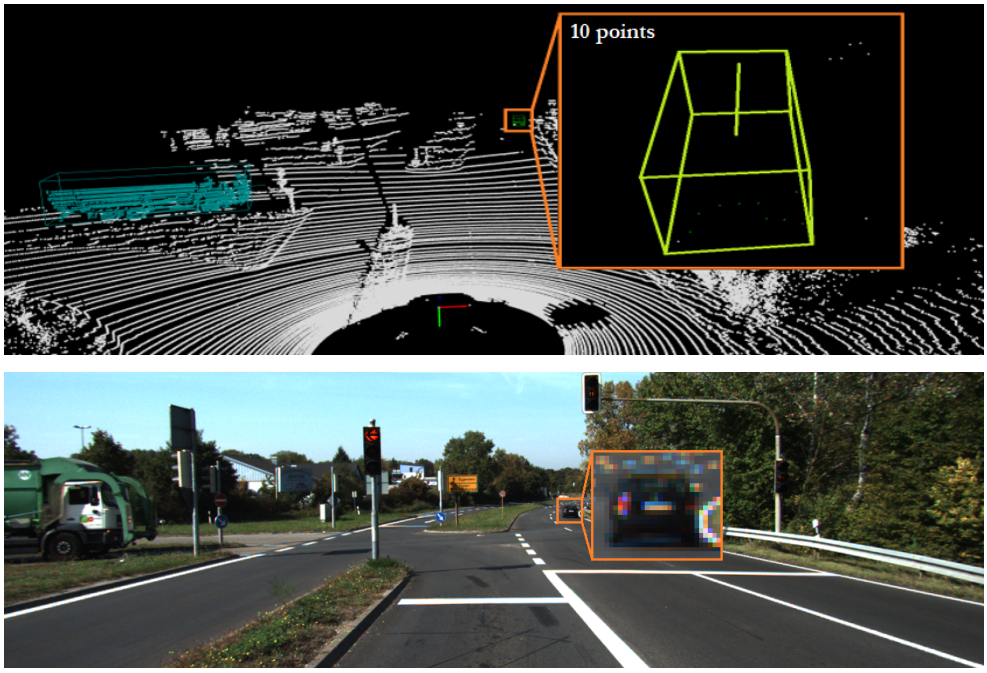



Figure 2.3: Example of a sparse point cloud

task of identifying and localizing objects of interest in an image. Prior to the advent of deep learning, object detection was done by localizing distinct manually engineered feature points. The need for a well-versed computer vision engineer to craft features has been a big barrier for classic object detectors to break into mainstream applications. However, recent developments in convolutional neural networks, the explosion of data and the availability of affordable computing have escalated the adoption of object detection algorithms in a variety of products.

Despite all the progress, most object detectors [29, 74, 61, 93] produce 2D bounding boxes which are inadequate for a number of domains such as robotics, augmented reality and autonomous vehicles. Hence, 3D object detection is a new line of research involving the estimation of the position and orientation of objects in 3D space. Compared to the regular object detection, 3D object detection raises several challenges; the most evident being the introduction of additional degrees of freedom. Furthermore, inferring depth from monocular images is ambiguous and often leads to large margins of error.

Image segmentation is another important computer vision task. It is the process of assigning each pixel in an image to one of the predetermined classes in order to partition

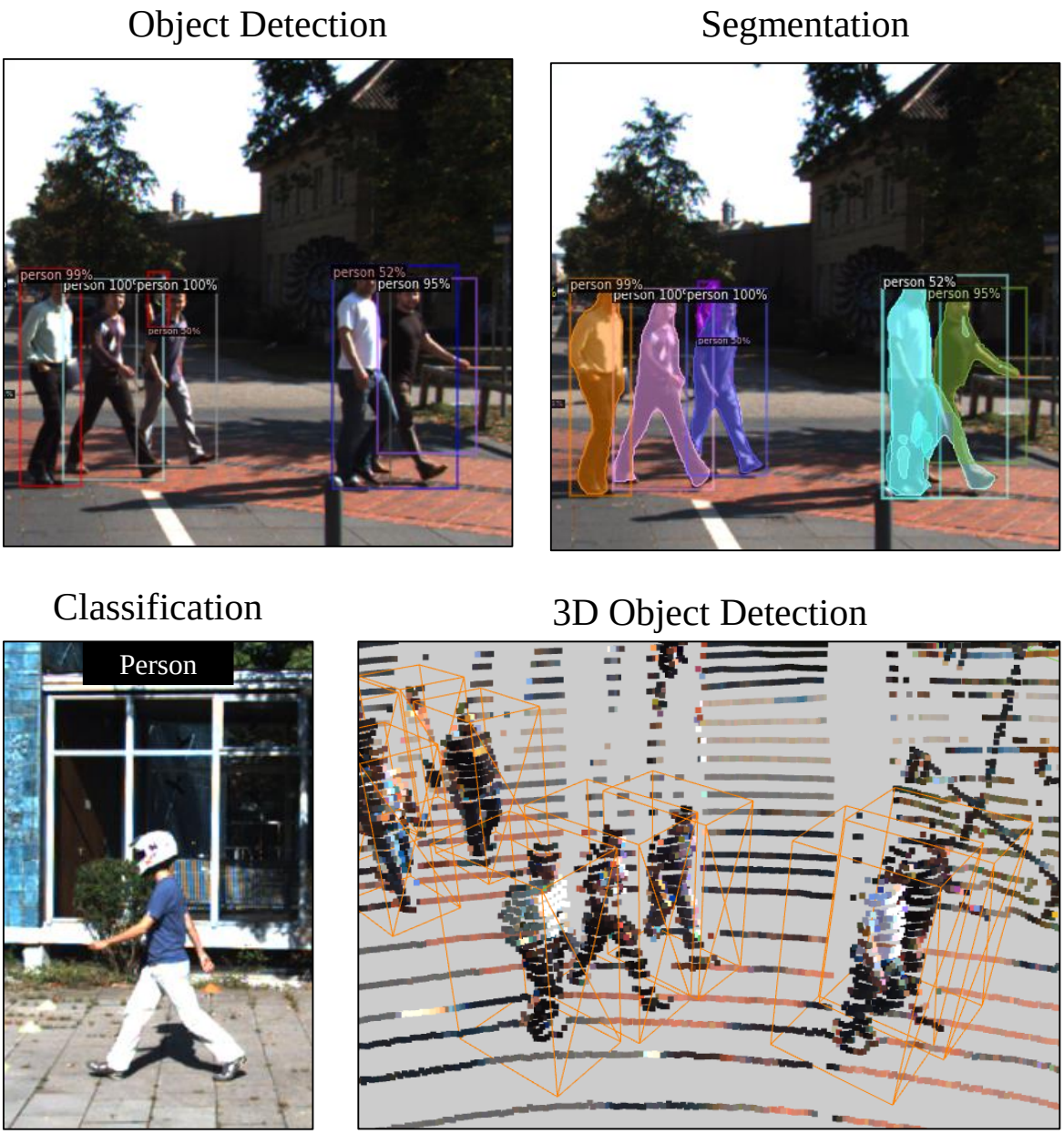



Figure 2.4: Computer Vision Tasks (From Top to Bottom): Object Detection, Segmentation, Classification and 3D Object Detection

the image into multiple semantic clusters. In contrast to detection, it provides a higher level of granularity. Segmentation techniques are further subdivided into semantic segmentation and instance segmentation. The tasks presented above have been visually summarized in Figure 2.4.

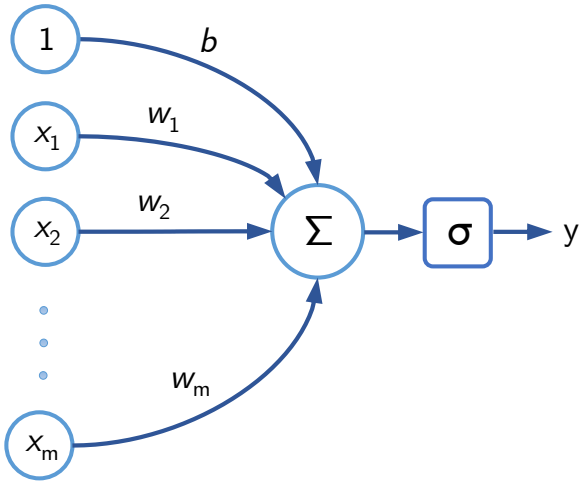
## 2.4 Neural Networks

Neural networks lie at the core of the current AI resurgence. Early works dating back to the 1950s where loosely inspired by biological neurons. Recently, neural networks have become the go-to method for a variety of *intelligence* problems such as natural language understand, medical diagnosis, stock price forecasting, etc . Nonetheless, they have not had the current level of attention from the research community and the industry until the famous AlexNet [46] won the ImageNet Challenge [79] in 2012. The two main reasons hampering the adoption of neural networks were the lack of sufficient training data and limitations in computing power.

### 2.4.1 Feedforward Neural Networks

The *perceptron*, illustrated in Figure 2.5, is the basic building block of all neural networks. It is a simple function that maps an input $X \in \mathbb{R}^m$ to an output $Y \in \mathbb{R}$ and is parameterized by a weight vector $W$ and a bias $b$. For the sake of simplicity, several textbooks absorb the bias into the weight vector and append a constant '1' to the input. The intermediate output of the function is computed by summing the elementwise multiplication of the input and the weight vector.

An activation function $\sigma$ is applied after summation to introduce non-linearity to the model. Some of the widely used activations along with their corresponding mathematical formulas are shown in Figure 2.6. Rectified Linear Unit (ReLU) [65] is preferred over sigmoid and tanh for most computer vision applications since the former is computationally cheaper to execute. Moreover, saturating gradients exhibited in sigmoid and tanh lead to a slower convergence.




$$y = f(x, [W, b]) = \sum_{i=1}^{m} w_i \cdot x_i + b$$

Figure 2.5: Perceptron

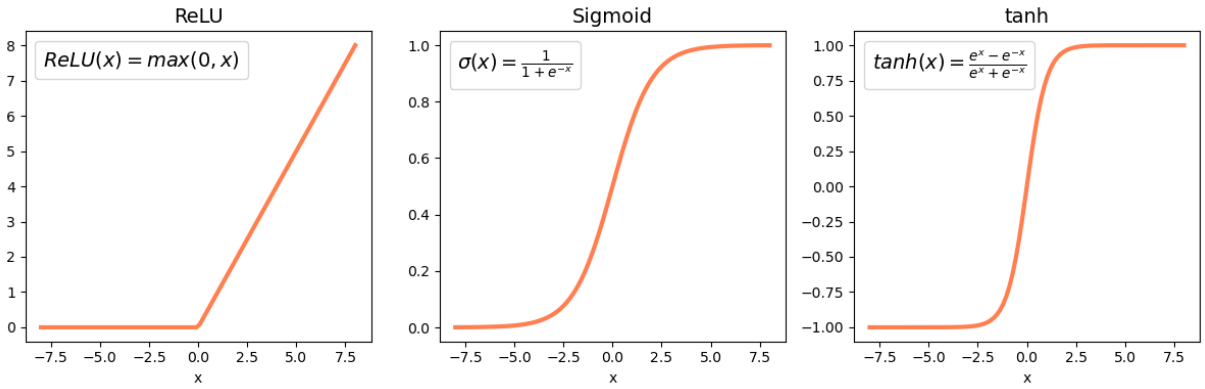



Figure 2.6: Three of the most commonly used activation functions

Multiple perceptrons can be chained together to form a *multilayer perceptron* (MLP) or a *fully connected feedforward network* (FCN). Compared to a single perceptron, an MLP is capable of learning more complex input-output mappings due to the higher number of parameters and nonlinear operations. The recent surge of interest in machine learning can be mainly attributed to the invention of deeper and wider networks.

### 2.4.2 Convolutional Neural Networks

Numerous configurations of artificial neural networks have been proposed over the years to address different challenges. Among these, convolutional neural networks (CNNs) are currently the most effective variant for vision related tasks such as classification, detection and segmentation. CNNs typically contain a convolution layer, a pooling layer and an optional upsampling layer.

#### Convolution Layer

Convolution is a linear mathematical operation on two functions $f$ and $g$ that captures the amount of overlap of $f$ as it is shifted over $g$ or vice versa. The convolution of two continuous functions is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.1}$$

The input of a CNN (*i.e.* a digital image) is usually discretized into an evenly divided grid. In that manner, we write the discrete convolution on two functions $f$ and $g$ defined on $\mathbb{Z}$, where $\mathbb{Z}$ denotes the set of all integers, as:

$$(f * g)(k) = \sum_{i \in \mathbb{Z}} f(i)g(k-i) \tag{2.2}$$

CNNs are made of up several convolution layers. As shown in Figure 2.7, $f$ is often referred to as the *image* or *input feature map*, $g$ as the *kernel* or *filter* and $(f * g)$ as the

*output feature map*. All of these variables are usually multidimensional arrays. Thus, we extend the previous definition to two dimensions:

$$(f * g)(x, y) = \sum_{x' \in \mathbb{Z}} \sum_{y' \in \mathbb{Z}} f(x', y') g(x - x', y - y') \tag{2.3}$$

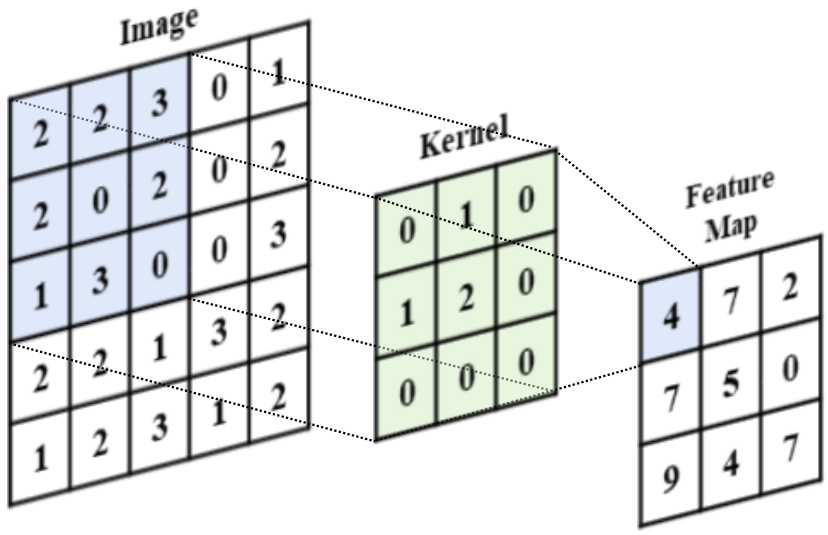



Figure 2.7: 2D convolution of a $5 \times 5$ image with a $3 \times 3$ filter

In practice, deep learning researchers do not flip the kernel when sliding it across the feature map. The formal mathematical term for such an operation is *correlation*.

CNNs provide several advantages over FCNs. In a fully connected layer, each neuron in the input is connected to every neuron in the output. Convolutional layers, on the other hand, exploit the spatial correlation of the data and compute the output of each neuron using a few neighbouring neurons from the previous layer. This results in a substantial decrease in the number of parameters. On top of the memory and compute efficiency, the smaller number of parameters reduces the likelihood of overfitting. Moreover, CNNs are also translation invariant — not affected by changes in position. Nonetheless, a convolutional layer that uses a kernel as big as the input image is equivalent to a fully connected layer.

### Pooling Layers

Pooling layers are usually placed in between convolution layers to reduce the spatial dimensions of feature maps. This operation decreases the number of parameters and thereby helps the network learn a more abstract representation of the input. It also reduces the training time and memory requirements. Max pooling and average pooling, illustrated in Figure 2.8, are widely adopted pooling layers. In comparison to the convolution operation, both layers are incapable of learning due to the lack of trainable parameters. Therefore, it has been common practice to use strided convolutions (stride $\geqslant 2$) instead and enhance the models expressive power.

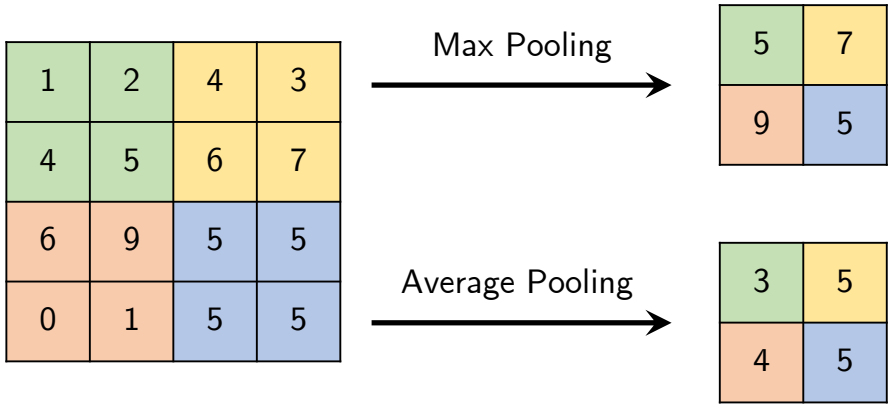



Figure 2.8: Max and average pooling with window size of 2 x 2 and stride 2

### Upsampling Layer

Contemporary deep learning models, for example segmentation and detection networks, need to upsample feature maps to fuse information from multiple abstraction levels. Nearest neighbour, bilinear interpolation, max-unpooling and transpose convolution are some of the most widely used techniques for upsampling. In nearest neighbour, the input pixel is expanded by duplicating across neighbouring pixels while bilinear interpolation computes a weighted average based on the distance from adjacent pixels. Unlike these fixed methods, transpose convolution (sometimes referred to as *deconvolution* amongst the deep learning community) uses learnable parameters and thereby yields a more flexible model. Yet, transpose convolution is known to suffer from the checkerboard effect [69].

### 2.4.3 Loss Functions and Optimization

The similarity between the output of a model $\hat{y}$ and the expected value $y$ is quantified by a task dependent metric called the loss or cost function. Cross entropy (Equation 2.4) is usually the first choice for classification while $L_1$ (Equation 2.5) and $L_2$ (Equation 2.6) losses are used for regression.

$$\mathrm{H}(y, \hat{y}) = -\frac{1}{N}\left(\sum_{i=1}^{N} y_i \cdot \log \hat{y}_i\right) \tag{2.4}$$

$$\mathrm{MAE}(y, \hat{y}) = -\frac{1}{N}\left(\sum_{i=1}^{N} |y_i - \hat{y}_i|\right) \tag{2.5}$$

$$\mathrm{MSE}(y, \hat{y}) = -\frac{1}{N}\left(\sum_{i=1}^{N} (y_i - \hat{y}_i)^2\right) \tag{2.6}$$

In supervised machine learning, optimization (popularly referred to as training or learning) is the process of minimizing the loss function by adjusting the parameters of the system. Even though several optimization methods have been explored in the past, gradient-based optimizations, particularly gradient descent, are the most pragmatic option for training deep neural networks. Gradient descent is an iterative approach of finding the local minimum of the cost function by taking small steps. The gradient $\nabla\mathcal{L}$, computed as the partial derivative of the loss with respect to each of the network parameters, provides the magnitude and direction of the next step. The actual size of the step is linearly scaled by a hyperparameter called the learning rate $\alpha$. A smaller learning rate will take longer to converge but guarantees, at the very least, a local minimum. By the same token, a larger step size will speed up training but might oscillate and never attain the minimum point.

### 2.4.4 Regularization

Three different models that learn points randomly sampled from a sinusoidal wave are illustrated in Figure 2.9. The first model is underfitting because it has failed to capture an

acceptable representation of the underlying training data. In contrast, the last model overfits the training data — 'memorized' every instance but has not learned a useful structure. Deep neural networks are over-parameterized and hence prone to overfitting. If not dealt with, it limits their use in out-of-sample data. Regularization is the technique of reducing the performance gap between the training and test set by introducing constraints to the model, synthesizing more training samples, etc.

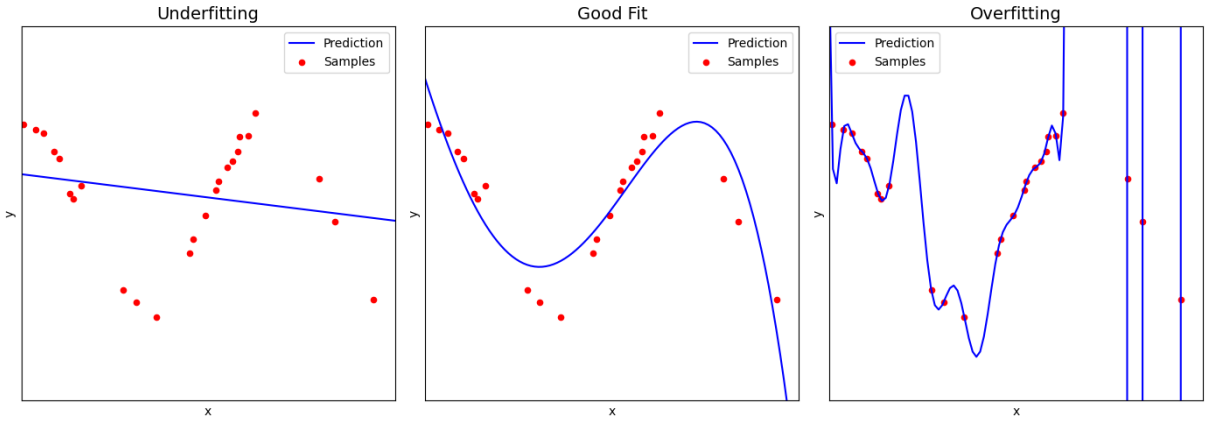



Figure 2.9: Underfitting vs Overfitting

### Augmentation

The easiest and most straightforward method to prevent overfitting is to add more data. Nonetheless, collecting, cleaning and labelling training data is an expensive and time consuming undertaking. Furthermore, data in domains such as medical imaging is very scarce. Therefore, researchers and engineers generate new samples by applying random transformations on available data. This procedure is known as data augmentation. When augmenting a dataset, care must be taken to verify that the generated samples are reasonable candidates compared to the original distribution. Rotation, translation, scaling, flipping, cropping, color adjustment, etc. are some of the most common image augmentation techniques. Some of these transformations can be easily adapted to point clouds.

#### Weight Regularization [47]

Weight regularization constrains the magnitude of the weights of the network by introducing additional terms to the total loss. In the case of L2 regularization, the sum of the squares of each weight is added to the loss. This encourages the network to learn several smaller weights instead of few larger ones. Likewise, for L1 regularization, the sum of the absolute values of each weight is added instead. Models trained with L1 regularization have sparser weight matrices.

#### Dropout [88]

Dropout is another commonly used regularization technique that reduces co-adaptation of neurons by randomly suppressing a fraction of them. It prevents the model from relying on a single neuron and encourages it to form more connections. It can also be regarded as training multiple instances of the same model concurrently, and indirectly acquiring performance gain observed in ensembles [34]. Nevertheless, dropout is less effective in convolutional layers due to information leak through neighbouring units. DropBlock [28] patches this by dropping adjacent pixels together.

#### Early Stopping

Early stopping is a simple regularization method that is implemented with no modification to the neural network. The model is trained as usual and the training procedure is halted when the validation loss either increases or plateaus for a predefined time interval.

## 2.5 Conclusion

This chapter summarized the self-driving car software stack, sensors and common tasks in computer vision. We also presented the key components and conventional practices of training a neural network.

# Chapter 3

# Related Work

Present-day 3D object detectors are built upon ideas developed during 2D object detection research. Hence, we begin this chapter by summarizing fundamental principles in the 2D object detection literature. We then present an in-depth explanation and critical analysis of recent 3D object detectors.

## 3.1 2D Object Detectors

The problem of object detection can be formally formulated as the task of locating objects of interest in an image and categorizing them as one of $n$ predefined classes. Compared to classification, object detection is more challenging due to the one-to-many mapping (*i.e.* a single image can contain zero or more instances).

### 3.1.1 Classical Object Detectors

Prior to the wide acceptance of deep learning, object detection was achieved through the localization of distinct manually engineered features and shallow classifiers. The most widely used approach was to slide a multi-scale window across the entire image and classify each crop. The whole pipeline can be dissected into four steps: *region selection*, *feature extraction*, *classification* and *postprocessing*.

As mentioned earlier, regions are proposed by placing a multi-scale window at evenly spaced intervals. This approach is computationally expensive due to the large number of redundant calculations resulting from overlapping proposals. Afterwards, algorithms such as Histograms of Gradients (HOG) [18], Scale Invariant Feature Transforms (SIFT) [62] or Haar-like [56] are used to extract features. These features are assumed to be be high-level semantic representations of the input crop that are invariant to orientation, color and illumination of the object. In practice, it is challenging if not impossible to design robust feature descriptors for the myriad of objects present around us. Next, a classical classifier like a Support Vector Machines (SVM) [16] or a Deformable Part-based Model (DPM) [24] is used to classify the feature vector. At last, redundant boxes are cleaned up by greedy non-maximum suppression.

The Viola-Jones framework [42] is one of the most successful face detectors to be widely used in consumer electronics such as digital cameras. Nonetheless, object detection is yet another area where deep learning methods have made big strides and out-shined their classic predecessors.

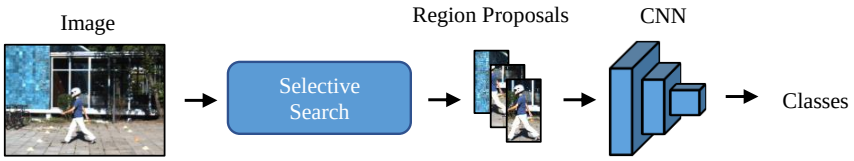
### 3.1.2 Region Based (Two-Stage) Methods

Region based or Two-Stage Object Detectors employ a two-step process where the first stage proposes potential regions that are highly likely to contain an object and the second stage classifies and optionally refines the proposals. Most notably, the RCNN series [30, 29, 77] have brought about profound changes and are reviewed below.

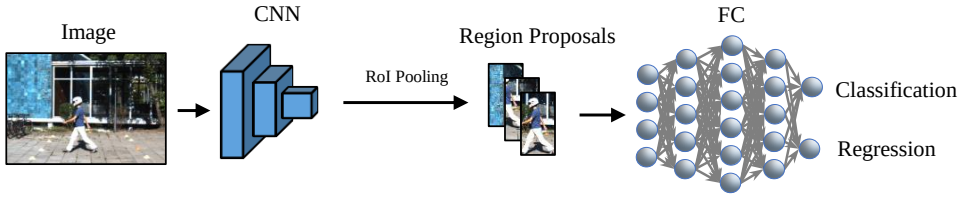
RCNN [30] uses selective search, a bottom-up approach for grouping pixels, to propose 2000 regions per image. Each region proposal is cropped, warped and passed through a pre-trained CNN for feature extraction and the resulting vectors are classified using an SVM. This method is very slow because features are extracted separately for each proposal. Fast RCNN [29] rectified this drawback by extracting the feature map first and then projecting the proposed regions onto it. This significantly reduced the runtime by sharing the computation required for feature extraction among regions. Moreover, Fast RCNN redefined the second stage by adding a regression task on top of the classification. Despite these im-

provement, selective search impeded Fast RCNN from running in real time. Thus, Faster RCNN [77] replaced selective search with a compact and fully convolutional network called region proposal network (RPN). As a result, the authors were able to train the network in an end-to-end manner. The architectures of all three methods are exhibited in Figure 3.1.

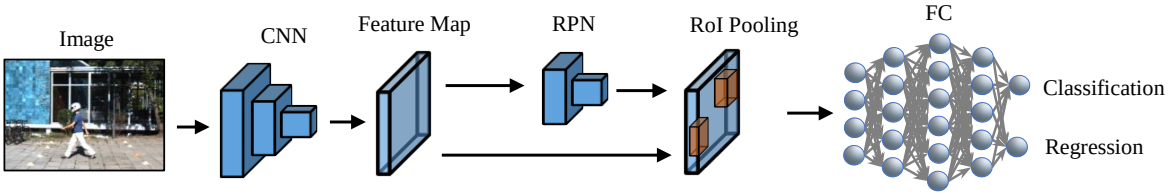
The multitask loss in Faster RCNN creates a conundrum between the classification head which is translation-invariant and the regression head which is translation-variant. That is to say, moving the object will alter its bounding box (regression targets) but not its class. To remedy this, the last convolution layer of R-FCN [17] produces $k^2$ position-sensitive score maps with a fixed grid of $k \times k$.




(a) RCNN




(b) Fast RCNN




(c) Faster RCNN

Figure 3.1: RCNN Series

### 3.1.3 Single-Stage Methods

Single-Stage detectors are more recent developments that estimate bounding boxes in one forward pass. Overfeat [80] is one of the pioneering works to merge the classification and

localization task into one model. Their approach is based on a multi-scale sliding window that directly predicts bounding boxes from the output feature map. Nonetheless, OverFeat has been succeeded by more accurate models like You Only Look Once (YOLO) [74, 75, 76] and Single Shot MultiBox Detector (SSD) [61]. YOLO and SSD (See Figure 3.2) are based on similar principles. They divide the input image into an $S \times S$ grid. Each cell in the grid is associated with one or more predefined priors called default boxes or anchors depending on the literature. The input image is passed through a series of convolutional blocks to output an $S \times S \times N$ tensor. The third dimension ($N$) is dependent on the number of classes and anchors. Even though YOLO and SSD highly resemble each other, the latter can detect objects with varying sizes because of its multi-scale detection head. Moreover, YOLO uses DarkNet [74] while SSD uses VGG16 [86] for feature extraction.

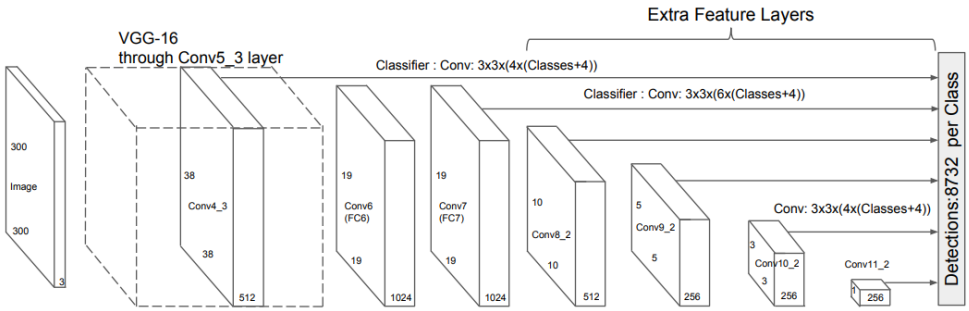



Figure 3.2: SSD: Single Shot MultiBox Detector (reprinted by permission from [61])

Initially, region based detectors had superior performance while single shot detectors yielded higher frame rates. However, recent advances in both areas have narrowed the gap. Two of the most important breakthroughs in single stage object detectors are covered below. See Figure 3.3 for a summary of accuracy vs speed and refer to [39] for a rigorous comparison.

**Feature Pyramid Network [57]** Objects come in different shapes and sizes. In particular, single stage methods struggle with detecting small objects. Inspired by image pyramids [3], Lin *et al.* [57] generate multi-scale feature maps by fusing information from intermediate feature maps. Although SSD [61] makes predictions at multiple levels, high level features are not propagated to lower layers. Refer to Figure 3.4 for a visual comparison of the two approaches. Instead of giving equal importance to all feature maps, BiFPN

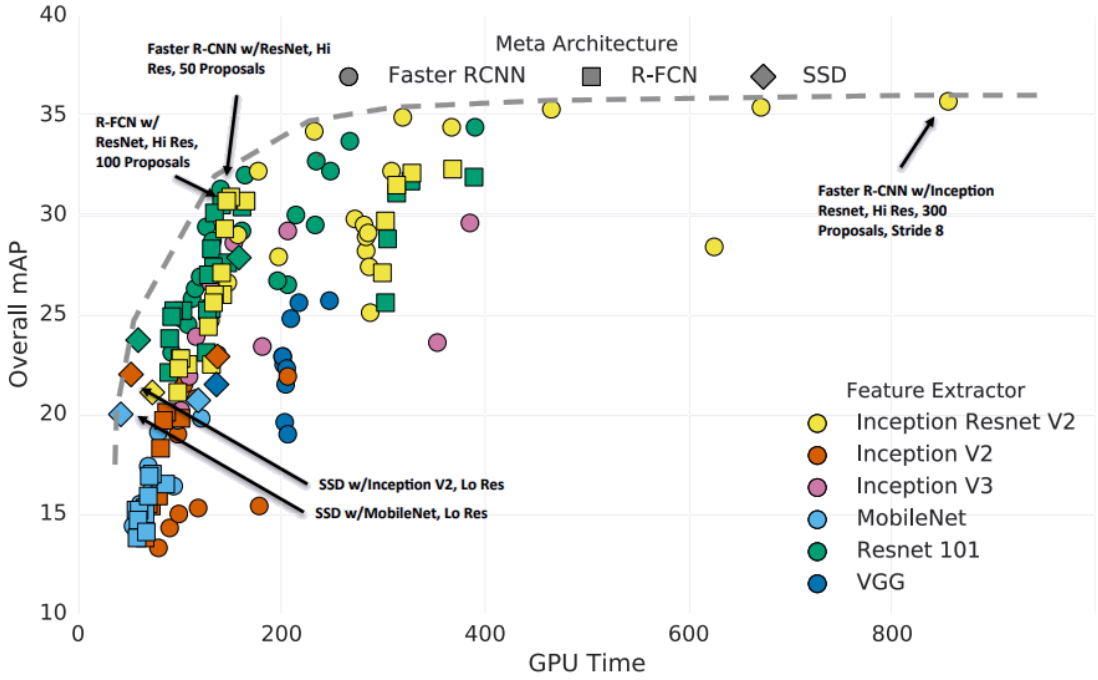



Figure 3.3: Speed/Accuracy trade-offs for modern convolutional object detectors (reprinted by permission from [39])

[92] fuses maps according to a learned weight matrix. Besides detection, FPNs have been shown to be useful in segmentation networks [44].

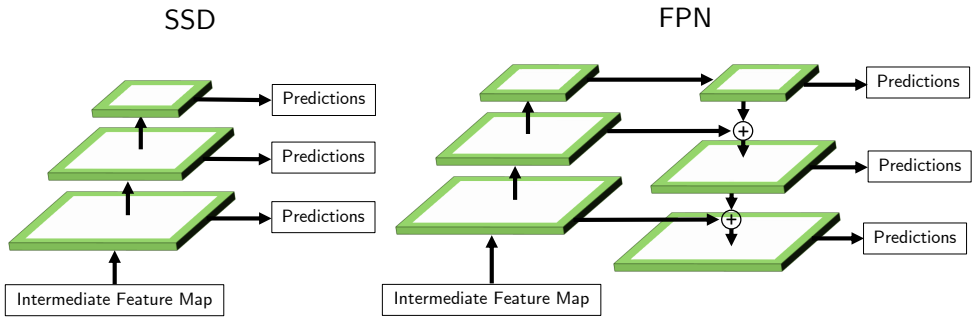



Figure 3.4: Comparing the feature level fusion of SSD [61] and FPN [57]

**Focal Loss** [59] Extreme background-foreground class imbalance is one of the main causes for a lower performance in single stage methods. In [59], the authors incorporated a modulating term to the regular cross entropy to diminish the contribution of easy examples (See Figure 3.5). RetinaNet [59], by combining FPN with focal loss, was able to outperform

most of the two stage detectors including Faster RCNN.

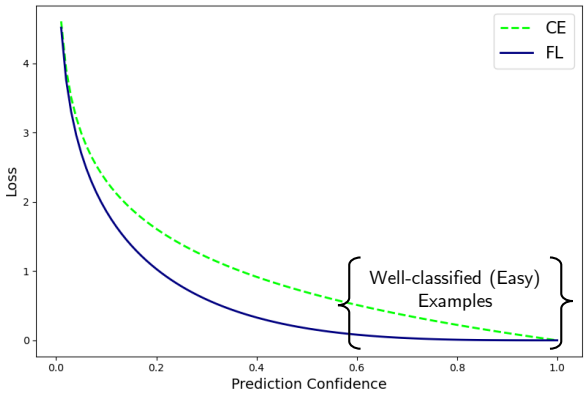



Figure 3.5: Focal Loss - the additional factor forces the network to put less emphasis on well-classified examples by down-weighting their contribution towards the total loss.

## 3.2 3D Object Detectors

Based on the modality in use, 3D object detectors can be grouped into three main categories: *3D Object Detection from Point Cloud*, *3D Object Detection from Images* and *Fusion Methods*. These classes along with further subdivisions [1] and a few representative papers are shown in Figure 3.6. The following sections will cover each in detail.

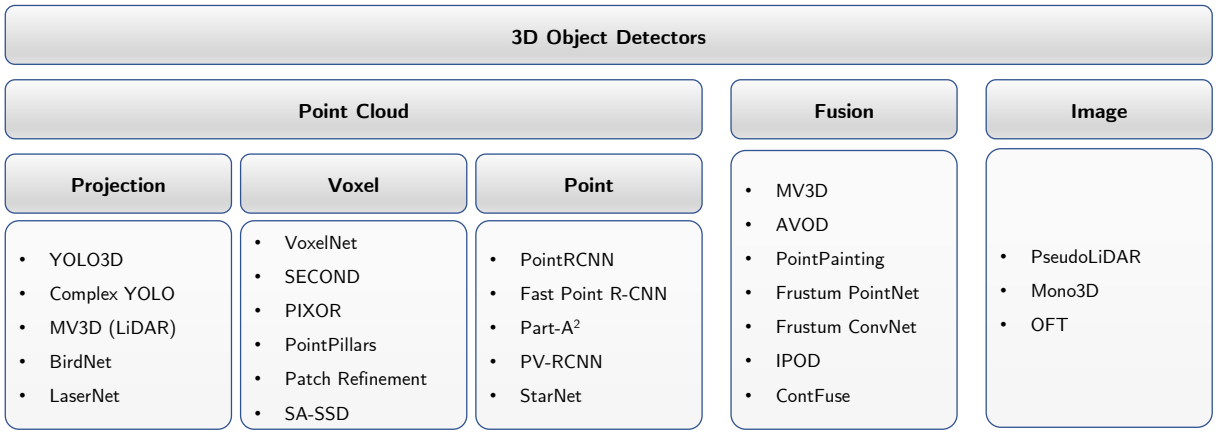



Figure 3.6: Family tree of 3D Object Detectors

[1] Some methods use a hybrid approach; thus do not cleanly fall into one category. In such instances, the novel aspect of the paper is considered for classification.

### 3.2.1 3D Object Detection from Point Cloud

Before we delve into 3D object detectors, it is mandatory to present PointNet [71], an integral part of several 3D detectors. PointNet is one of the first deep neural networks to work directly on raw point cloud. Prior to that, researchers either rendered the cloud into multiple views [89, 72] and classified with standard 2D CNNs or quantized the points into 3D voxels [101, 72] and processed it with a combination of 2D and 3D convolutions. As these transformations do not accurately capture all the information conveyed in the raw data, they had a negative impact on the performance of the classifier.

Given a point cloud containing $n$ points, the points can be arranged in $n!$ different ways. Subsequent processing must be invariant to the input representation. In order to achieve this, the authors [71] turned to symmetric functions — functions that yield the same output regardless of the order of the arguments. The PointNet network, illustrated in Figure 3.7, maps each of the $n$ input points from 3 dimensions to to a higher dimension using a series of shared MLPs. Once the points are transformed in to the higher-dimensional embedding space $\mathbb{R}^{1024}$, max pooling (a symmetric function) is used to generate a global feature vector. The generated vector summarizes the input point cloud and can be used for other tasks such as classification and regression. *Input transform* and *feature transform* are simple affine transforms that enhance the network's invariance towards permutation and geometric transformations. The details of these transformations are not relevant to the task at hand and will not be discussed here. One of the limitations in this work [71] is its inability to learn local structure. PointNet++ [73] rectifies this through a hierarchical approach of iteratively sampling and grouping points prior to passing through the MLP.

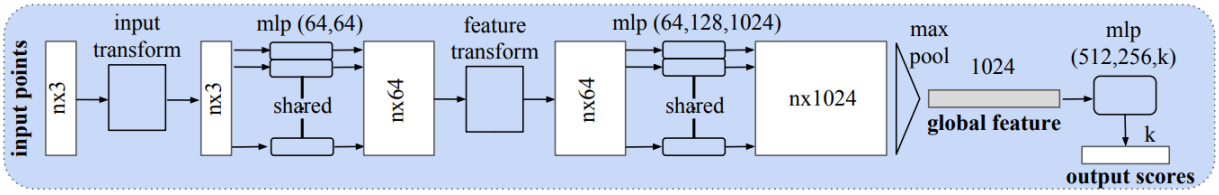
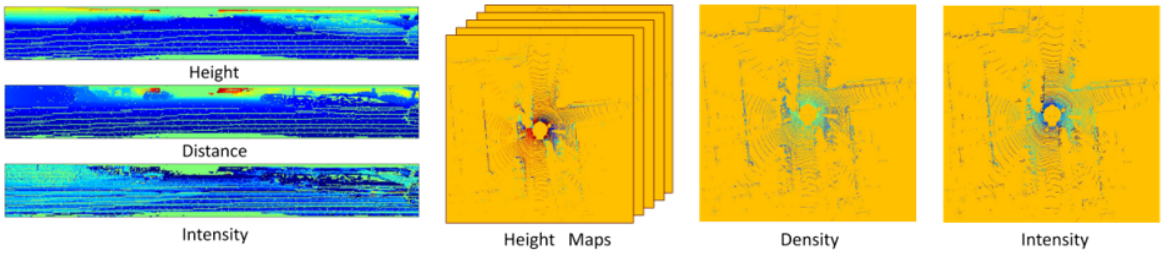



Figure 3.7: PointNet Architecture (reprinted by permission from [71])

**Projection based methods** project the point cloud into a 2D grid and reuse existing well-established 2D CNNs by incorporating additional regression targets for the new task.

A point cloud is natively captured in the range view (RV). Consequently, it is straightforward to construct an RV image by projecting the points into a cylinder and unraveling it into a rectangular image [53, 63]. The input image, shown in Figure 3.8a, captures various statistical features such as height, lateral distance and intensity. As this is an image, it can be processed using regular 2D architectures to produce objectness maps and bounding box coordinates. On the contrary, processing the point cloud in bird's eye view (BEV) instead of RV yields higher performance because the former preserves scale of objects and prevents occlusion under normal circumstances. Moreover, applying standard 2D convolutions to RV projections causes depth-blurring as shown in [96]. Thus, [8, 84, 7] use BEV projections (See Figure 3.8b) and are precursors to voxel based methods.




(a) Range view (b) Bird's eye view

Figure 3.8: Inputs to 2D CNN generated by projecting the point cloud (©2017 IEEE) [14]

**Voxel based methods** divide the point cloud into evenly spaced cuboids. Compared to the previous group, the voxelization process explicitly captures 3D information. PIXOR [103] encodes the input using a simple occupancy cuboid. The vertical information is stored along the channel axis while the spatial axes of the pseudo-image capture the remaining lateral dimensions. The output is also augmented with an additional channel that carries mean reflectance values. VoxelNet [109] and SECOND [102], on the other hand, divide the space into a grid of voxels and apply a Voxel Feature Encoding layer (a smaller form of PointNet [71]) to map the input into a higher dimension and more informative space. Next, they apply a set of 2D and 3D convolutions to classify and regress 3D bounding boxes. PointPillars [51] improves the runtime of [102] by removing the 3D convolutions, using an even smaller PointNet encoder and a single bin along the height axis in its voxel representation. Similarly, Patch Refinement [52] uses voxel representation at two differ-

ent resolutions. It stacks two convolutional networks where the second network refines estimates by fusing the information from the first network with higher resolution voxels. As it turns out, this method is not computationally efficient since it requires encoding two different voxel-based representations and detects objects in two stages. SA-SSD [35] implemented an auxiliary network that reverts intermediate convolutional feature maps into point-wise representations to help the backbone capture structural information in the point cloud. They also propose a part-sensitive warping operation that aligns confidence scores to predicted boxes.

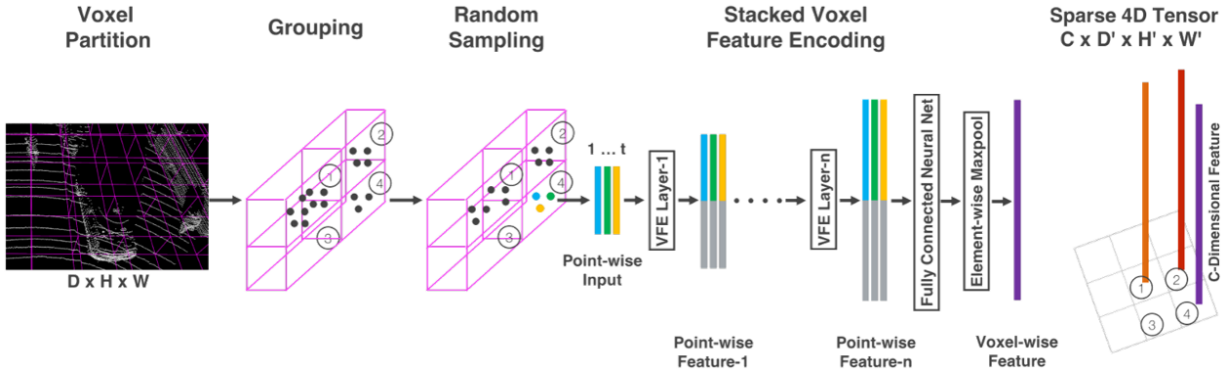



Figure 3.9: VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection (reprinted by permission from [109])

**Point based methods**, pioneered by Shi *et al.* [82], work directly on points rather than dividing the point cloud into a grid. These are mostly made up of two separate models where the first one proposes candidate centroids while the second stage refines and classifies them using a variant of PointNet [71]. Point RCNN [82] employs an expensive per-point segmentation to classify all the points as either foreground or background. Next, they crop the area around foreground points along with corresponding features learned from the previous stage for classifying and refining 3D bounding boxes. Fast Point R-CNN[15] replaces the first proposal generation stage with a faster VoxelRPN that resembles [109]. Part-$A^2$ [83] extends [82] with a part supervision task to improve performance on partially occluded and truncated objects. StarNet [66] proposed a computationally efficient method by adopting an engineered sampling strategy for proposing candidate regions. As opposed to the previous two, point based methods provide a flexible field of view that can be tuned to match the object of interest's dimensions.

### 3.2.2 Multimodal 3D Object Detection

Even though point clouds convey accurate 3D information, they suffer from sparsity, low resolution and lack of texture information. Hence, this sparks a new line of research of augmenting LiDAR based 3D detectors with additional modalities such as images. The BEV representation of a LiDAR scan is more favorable than its inherent form (RV) for reasons mentioned previously. Similarly, images are captured in RV; but it is not as straightforward to recover the BEV representation. The ideal solution in fusing the two modalities, yet to be seen, is to find a common representation that is as robust as BEV and easy to transform into for both.

MV3D [14], one of the earliest works on the KITTI dataset [26], generates RV and BEV projections to encode point cloud attributes such as density, height and intensity. Similar to Faster RCNN [77], a 2D CNN is used to generate 3D proposals from the LiDAR BEV map only. The proposals are projected into the LiDAR RV and RGB image. Next, all resulting features are cropped and aggregated by an RoI pooling layer. At last, the results are fused using a region-based deep fusion block to make predictions. The authors, in their experiments, show that deep fusion is marginally better than early and late fusion. In contrast, AVOD [48] uses a similar mechanism but generates proposals using features from both modalities. The full MV3D pipeline is illustrated in Figure 3.10

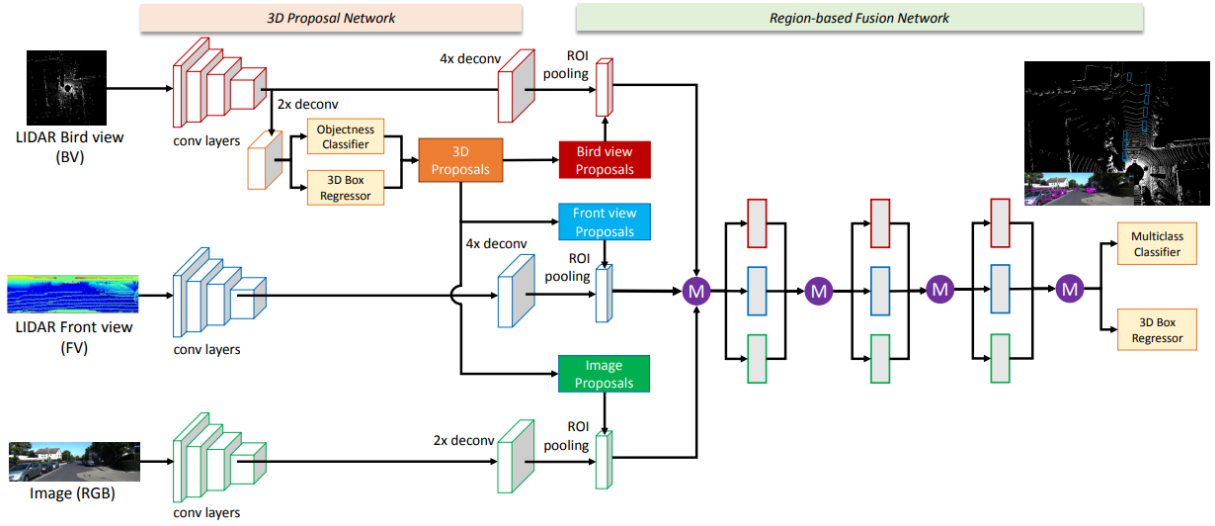



Figure 3.10: Multi-View 3D object detection network (©2017 IEEE) [14]

Frustum PointNet [70] and Frustum ConvNet [97] implement a sequential fusion scheme. Mature image-based 2D object detectors are used to reduce the search space by generating candidate frustums. The points enveloped by the proposed frustums are processed with PointNet [71] to regress 3D bounding boxes. IPOD [104], in contrast, uses semantic segmentation maps in place of an object detector. While augmenting the LiDAR data with images improves the precision, the recall is bounded by the performance of the first stage (image based detector). PointPainting [95], another sequential approach, decorates points with a class vector that is extracted from an image-only semantic segmentation network. The *painted* cloud is then processed by existing 3D object detectors, namely PointPillars[51] and Point RCNN [82]. Unlike other methods, were both modalities are processed in parallel, the sequential formulation introduces additional latency. The authors [95] suggest using segmentation masks from previous frames to minimize the delay on the basis that consecutive frames are mostly coherent.

Feature level fusion techniques run two concurrent streams for each modality and fuse intermediate feature maps. [98] proposed a new layer called *sparse non-homogeneous pooling* that transforms BEV into RV and vice versa. The former is exclusively used to make predictions. Even though the latter isn't used for predicting, it provides an auxiliary task and improves performance. MMF [54] combines 3D detection with 2D detection, depth completion and ground plane estimation to create a massive end to end trainable network. Nevertheless, both methods are prone to feature blurring due to the lack of a one-to-one correspondence between the fused feature maps [95]. ContFuse [55] attempted to mitigate this by using k-nearest neighbour and bilinear interpolation.

As of writing, the KITTI leaderboard is dominated by LiDAR only methods both in terms of speed and accuracy (See Figure 3.11). We hypothesize that the KITTI dataset is not big and diverse enough to learn a robust joint representation. Moreover, it is highly sensitive to trivial hyperparameters such as NMS threshold.

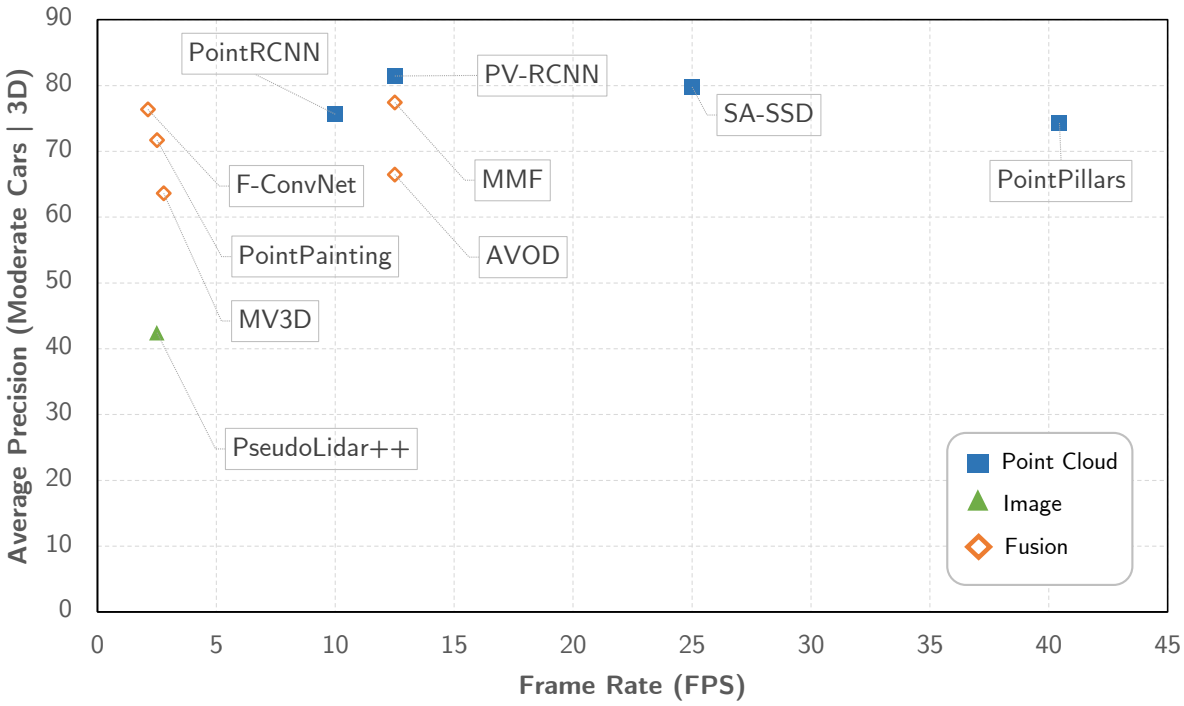



Figure 3.11: Average precision plotted against frame rate for select 3D object detectors

### 3.2.3 3D Object Detection from Images

The final class of detectors use image only. OFT [78] attempts to generate voxelized BEV features explicitly from the RGB input image. Objects as Points [107] extends its anchorless 2D object detection head to regress 7 additional values for 3D object detection. Pseudo-LiDAR [96, 105] argue that the main culprit in the low performance of image-based methods is the representation and not the quality of the data. Consequently, they propose a three-stage approach: estimating depth using [25, 13], followed by back-projection to lift the depth map into a pseudo point cloud and finally detection using preexisting 3D object detection methods. Nonetheless, a considerable gap still exists between algorithms using images and point clouds.

## 3.3 Conclusion

This chapter has attempted to provide an overview and critical comparison of modern object detectors. We started by explaining pre- deep learning methods of object detection

and identified that the feature engineering aspect was a big barrier in their wide application. Afterwards, we summarized single shot and region based 2D detectors. Next, we reviewed the 3D object literature and broadly categorized them into three based on the modality in use.

# Chapter 4

# Methodology

In this chapter, we present the methodology used in the context of this thesis. We first describe the storage and distribution of LiDAR data in the form of point clouds followed by an elaboration of the multimodal KITTI dataset [26]. Afterwards, we define evaluation metrics for quantifying and comparing the performance of our proposed models against other published results. The last section describes a post-processing stage employed by most object detectors for filtering duplicate detections.

## 4.1 Point Cloud

A point cloud is a convenient data structure used for processing, disseminating and storing data captured by a LiDAR. Mathematically, it is defined as an unordered set of 3D points $\mathbf{P} = \{p_i \mid i = 1, ..., n\}$ where $p_i$ is a vector of 3 (or possibly more) values and $n$ is the total number of points. The first three entries of a point $p_i$ indicate the spatial location in an orthogonal XYZ coordinate frame while the remaining values convey additional attributes such as reflection, color, etc. A point cloud extracted from the KITTI dataset [26] is rendered in Figure 4.1.

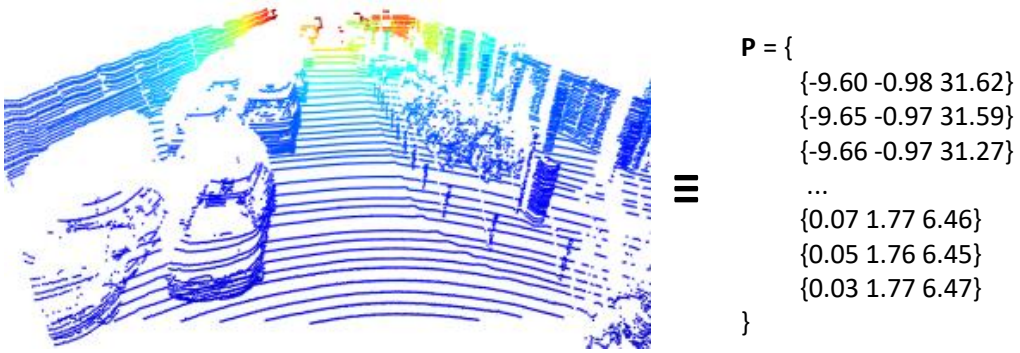



Figure 4.1: Left: Visual rendering of a LiDAR scan, Right: LiDAR scan represented as a point cloud

## 4.2 Dataset

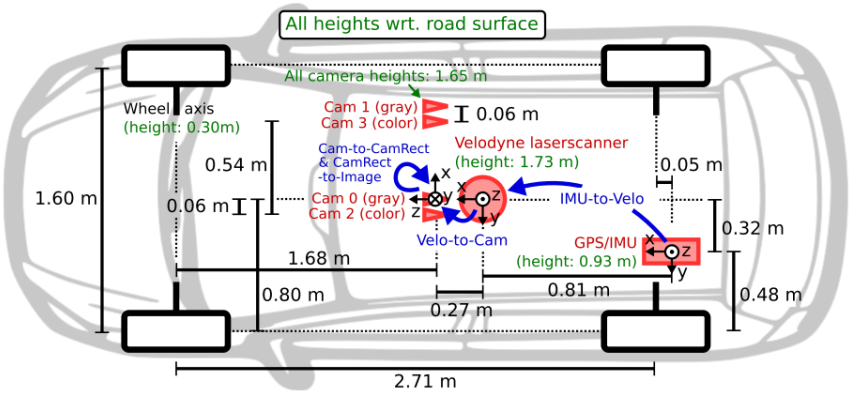
The wide availability of LiDARs has resulted in an explosion of multimodal datasets over the last few years. KITTI [26], Nuscenes [10] and Waymo [90] are some of the most prominent examples. Of these, the KITTI dataset has grown to become the standard benchmark for numerous tasks related to autonomous vehilces; monocular and stereo object detection, multi-object tracking, depth completion, instance and semantic segmentation of LiDAR to name a few. Consequently, all of our models where trained and evaluated on the pioneering KITTI dataset. Using this dataset permits us to compare our performance against other frameworks on both BEV and 3D object detection benchmarks.

The dataset contains diverse driving scenes collected at urban streets, rural neighbourhoods and highways from Karlsruhe, Germany. The data acquisition vehicle is shown in Figure 4.2a. An entry in the KITTI dataset consists of a pair of color images, a pair of grayscale images, a 360° point cloud collected using a 64 beam LiDAR, calibration matrices and most importantly manually annotated 3D bounding boxes.

In machine learning problems, datasets are split into training and validation sets. The training set is used to update the weights of the model (*i.e.* the model will have access to this subset only). The validation set, in contrast, is hidden from the model and used to evaluate its performance after training. Randomly splitting a dataset where individual samples are closely related will leak information from the validation set into the training

process. This occurrence is referred to as *group leakage* and will often lead to an overestimation of the model's generalization capability. As the authors of the KITTI dataset have not released an official training-validation split, we follow the common practice [14, 64] and split the training set into 3,712 training samples and 3,769 validation samples. This division guarantees that the dataset was divided on a scene basis and that frames from the same scene are not shared between the splits.


(a) Data acquisition platform




(b) Sensor configurations with respect to the ego vehicle

Figure 4.2: The KITTI vehicle is equipped with two color and two grayscale cameras, a 360° mechanical 3D laser scanner and a GPS/IMU inertial navigation system (reprinted by permission from [26])

A point in the KITTI point cloud is encoded as tuple of dimension 4 constructed by concatenating the point's Cartesian coordinate in LiDAR frame and its intensity. In

contrast to images which contain a fixed number of pixels, the number of points in a point cloud depends on the scene. In the case of KITTI, a single scan contains 120K points on average. However, filtering points outside the camera's field of view brings it down to 20K. The point cloud, in its raw form, is provided in the LiDAR coordinate frame while objects are annotated in rectified camera coordinate frame. Thus, it is necessary to consolidate these frames before we begin training. We transform every point $p$ in the point cloud as shown in Equation 4.1. The positions and orientations of both sensors with respect to the ego vehicle are shown in Figure 4.2b. In the rectified camera coordinate frame, the positive z-axis points in the motion of the ego vehicle while the x-axis points to the right. The rectified camera coordinate is the default choice for the remaining parts of this thesis.

$$\bar{\mathrm{p}}^{rect} = \mathrm{H}^{rect}_{cam_2} \cdot \mathrm{H}^{cam_2}_{velo} \cdot \bar{\mathrm{p}}^{velo} \tag{4.1}$$

where $\bar{\mathrm{p}}^{A}$ represents a point (in homogeneous coordinates) in *Frame A* and $\mathrm{H}^{B}_{A}$ is a $4 \times 4$ homogeneous transformation matrix that maps points from *Frame A* to *Frame B*.

The KITTI dataset splits all 3D ground truth boxes into *easy*, *moderate* and *hard* difficulty levels on the basis of occlusion, truncation and size of projection in the image. The specifics of each level can be found in Table 4.1. We use the initials (**E**, **M**, **H**) when reporting results in subsequent chapters.

Table 4.1: KITTI difficulty levels

| **Difficulty** | **Max. Occlusion** | **Max. Truncation** | **Min. 2D Box Height** |
|---|---|---|---|
| **Easy (E)** | Fully visible | 15% | 40 px |
| **Moderate (M)** | Partly occluded | 30% | 25 px |
| **Hard (H)** | Difficult to see | 50% | 25 px |

## 4.3 Metrics

### 4.3.1 Intersection over Union

The Intersection over Union (IoU) or the Jaccard Index measures the amount of overlap between two boxes. It is not directly used as an object detection metric, but is essential in computing average precision. The IoU between two 2D bounding boxes $B_1$ and $B_2$ is intuitively shown in Figure 4.3 and mathematically defined as:

$$\text{IoU}(B_1, B_2) = \frac{\text{area}(B_1 \cap B_2)}{\text{area}(B_1 \cup B_2)} = \frac{\text{area}(B_1 \cap B_2)}{\text{area}(B_1) + \text{area}(B_2) - \text{area}(B_1 \cap B_2)} \tag{4.2}$$

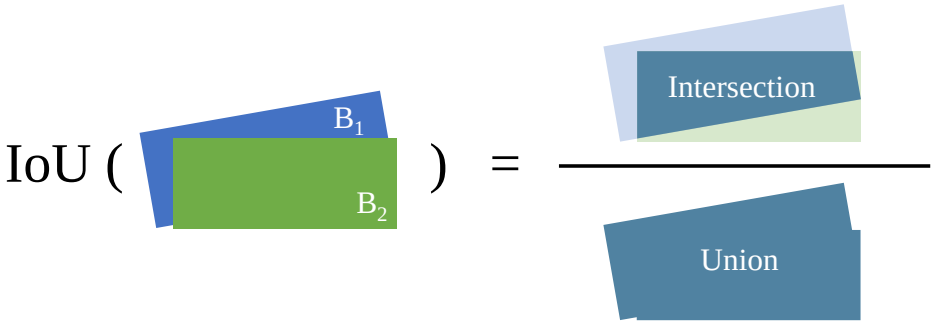



Figure 4.3: Intersection over union of two boxes

In the context of object detection, predictions are classified as *true positive* (TP) or *false positive* (FP) based on their IoU with a ground truth box. If a prediction has an IoU (with a ground truth box) greater than a preset *IoU threshold*, it qualifies as a TP, else it is assumed to be a FP. Similarly, if the maximum IoU between a ground truth box and all other predictions is less than the IoU threshold, the ground truth box is considered to be a *false negative* (FN). A technical detail worth mentioning is that a ground truth box can be matched to one prediction only. In occasional circumstances where more than one detection transcends the IoU threshold, the one with the highest overlap is marked as a TP while remaining detections become FPs.

### 4.3.2 Precision-Recall Curve

Precision and recall are performance measures commonly used for class imbalanced problems such as object detection. Precision is an indication of the quality of predictions and defined as the ratio of correct positive predictions to total predictions. Recall, on the other hand, is the fraction of correct positive predictions to total positive instances. Mathematically, precision and recall are defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.3}$$

where TP, FP and FN represent the number of true positives, false positives and false negatives respectively.

Every predicted bounding box has a confidence score attached to it. Changing the confidence threshold of the model will affect the precision and recall. Thus, we plot the precision-recall pairs at different confidence thresholds to visually inspect the trade-off between the two metrics. The resulting plot is called a *precision-recall (PR) curve*. An ideal PR curve, illustrated in Figure 4.4, has a perfect precision for all recall points.

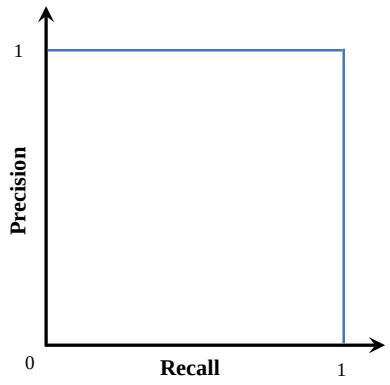



Figure 4.4: Ideal precision-recall curve

### 4.3.3 Average Precision

Average precision (AP) was introduced in 1983 by Salton *et al.* [21] for evaluating information retrieval systems. It was later popularized amongst the computer vision community after its use in the PASCAL VOC Challenges [22, 23]. Intuitively, it can be seen as the average of several precisions corresponding to evenly spaced recall points. It it computed as the area under the PR curve. The general formula for AP is given as:

$$\text{AP} = \int_0^1 p(r)dr \tag{4.4}$$

where $p(r)$ is a function that returns the precision at recall $r$.

Instead of using the theoretical infinitesimally small recall differential $dr$, the official KITTI evaluation toolkit [26] along with other object detection benchmarks [22, 60, 49] approximate AP using 11 recall points. With the intent of reducing the noise caused by variations in the ranking of detections, they use an interpolated PR curve where the precision $p(r)$ at recall $r$ is set to the maximum precision for any recall point $\tilde{r} \geqslant r$. The perfect PR curve of Figure 4.4 would have an AP of 1.0. Mathematically $\text{AP}_{\text{R11}}$ is defined as:

$$\text{AP}_{\text{R11}} = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1.0\}} \max_{\tilde{r} \geqslant r} (p(\tilde{r})) \tag{4.5}$$

Following a major flaw in the 11-point recall AP used in KITTI, Simonelli *et al.* in [85] proposed to compute the average precision on 40 recall points and ignore the precision at $r = 0$. Both metrics are widely used within the current 3D object detection literature. Nevertheless, for consistency and fair comparison, we report all results (ours and other state-of-the-art) on $\text{AP}_{\text{R40}}$ defined in Equation 4.6.

$$\text{AP}_{\text{R40}} = \frac{1}{40} \sum_{r \in \{\frac{1}{40}, \frac{2}{40} .., 1.0\}} \max_{\tilde{r} \geq r} (p(\tilde{r})) \tag{4.6}$$

### 4.3.4 Maximum $F_1$ Score

Another widely used metric for classification and detection is the $F_1$ score. In practice, the confidence threshold is set to a constant value when deploying a model. Aghdam *et al.* in [5] showed that two hypothetical models can have different APs but the same practical performance (See Figure 4.5). This necessitates for an alternative metrics that summarizes the practical competency of an algorithm. Hence, we follow [5] and compute maximum $F_1$ score on each difficulty level $c \in \{\mathbf{E}, \mathbf{M}, \mathbf{H}\}$ as:

$$F_1^c = \max_t \left[ 2 \cdot \frac{p_c^t * r_c^t}{p_c^t + r_c^t} \right] \tag{4.7}$$

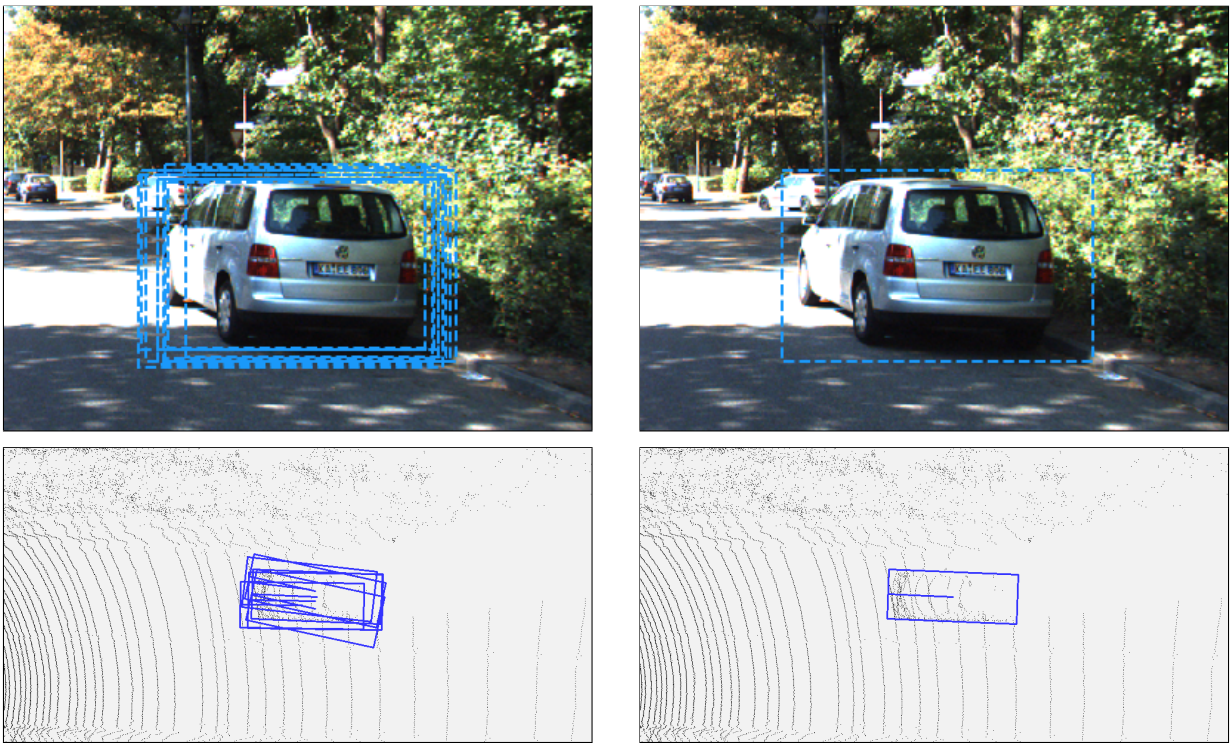
where $p_c^t$ and $r_c^t$ are precision and recall for level $c$ at confidence threshold $t$, respectively.


Figure 4.5: The maximum $F_1$ score for both hypothetical PR-curves is observed at their intersection. This yields different APs but identical practical performance. [5]

## 4.4 Non-maximum Suppression

Contemporary 2D and 3D object detection models produce multiple bounding boxes for the same ground truth box. As a result, an iterative post-processing step called non-maximum suppression (NMS) is applied to the final predictions in order to eliminate redundant bounding boxes. With the exception of the IoU calculation step, the same NMS procedure

can be applied to both 2D and 3D bounding boxes. For this work, we enhance the runtime and performance of the common greedy NMS with two simple modifications. (1) Computing the IoU between two 3D bounding boxes takes almost 300 times longer than computing the distance between two points. Thus, we replace the IoU based similarity kernel with a much faster distance kernel. The distance is calculated between the center of the box with the highest confidence and all other bounding boxes centers. (2) Our model predicts dense classification maps. Hence, during NMS, we filter out bounding boxes that are not surrounded by at least $T_n$ neighbours. Our procedure is summarized in Algorithm 1.



(a) Initial Predictions

(b) Filtered with NMS

Figure 4.6: Non-maximum Suppression

**Algorithm 1:** Distance based NMS

**Inputs :** List of detected 3D bounding boxes $\mathbf{B_i} = \{b_0, b_1, ..., b_n\}$

Distance Threshold $T_d$

Minimum Neighbour Count $T_n$

**Output:** List of filtered 3D bounding boxes $\mathbf{B_o} = \{b_0, b_1, ..., b_m\}$

$\mathbf{B_o} \leftarrow \{\}$

$\mathbf{B_i} \leftarrow \text{sort}(\mathbf{B_i})$

**while** $\mathbf{B_i} \neq \emptyset$ **do**

  $b_t \leftarrow \mathbf{B_i}[0]$

  $\mathbf{B_i} \leftarrow \mathbf{B_i} - b_t$

  $count \leftarrow 0$

  **for** $b$ *in* $\mathbf{B_i}$ **do**

    **if** *center-to-center-dist*$(b, b_i) \geqslant T_d$ **then**

      $count \leftarrow count + 1$

      **if** $count \geqslant T_n$ **then**

        $\mathbf{B_o} \leftarrow \mathbf{B_o} + b_t$

      **end**

    **end**

  **end**

**end**

## 4.5 Conclusion

This chapter provided a formal definition of the point cloud data structure as well as a brief introduction to the KITTI dataset [26]. Moreover, the metrics outlined in this chapter will be used to quantitatively assess the models presented in Chapter 5 and Chapter 6 and the post-processing stage will be used to suppress overlapping detections.

# Chapter 5

# FA3D: Fast and Accurate 3D Object Detection

This chapter presents our proposed fast and deep 3D detection network that utilizes sparse occupancy matrices for fast and compact representations of point clouds. At the time of writing, our model was the fastest 3D object detector reported on the KITTI benchmark running at 57.83 FPS on a high-end workstation. That makes it 43% faster than the fastest published work PointPillars [51]. Our gain in speed is attributed to an efficient input encoding that is multiple times faster to compute on a CPU and more compact to transfer to a GPU. More importantly, we do not improve the execution time by sacrificing the neural network's expressive power. A compressed version [19] of this chapter was published at the $15^{\text{th}}$ International Symposium on Visual Computing (ISVC) .

## 5.1 Input Representation

By design, conventional convolutional neural networks operate on discrete regularly sampled data. For example, color images are encoded as a 3 dimensional tensor of size $H \times W \times 3$. Each pixel in the image has a high correlation with all its neighboring pixels. On the contrary, a point cloud is an unordered set of points where location within the set has no spatial interpretation. Furthermore, permutation of the points does not alter

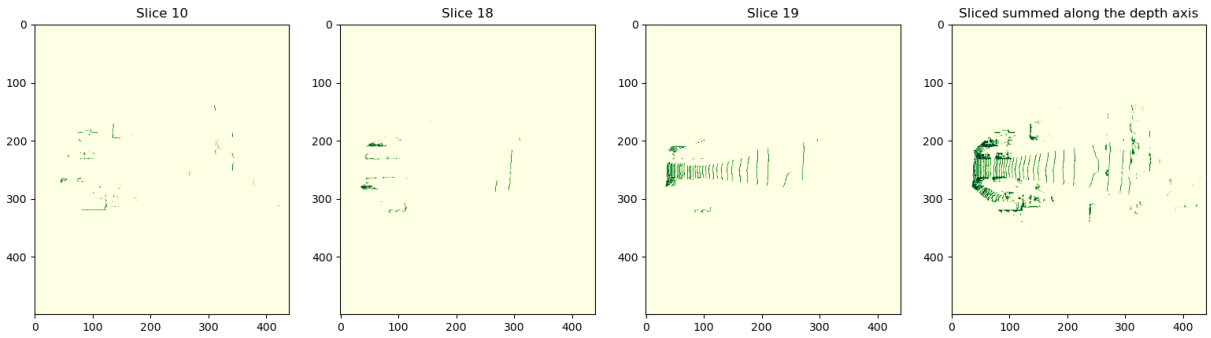



Figure 5.1: Occupancy Cuboid Slices

the captured information. These facts impede the use of standard convolutional neural networks on raw point cloud.

Having said that, we elaborate on the procedure followed to convert the raw point cloud into a uniform tensor that is readily processed by a CNN. Denoting the $i^{th}$ 3D point in a LiDAR point cloud using $p_i = (x_i, y_i, z_i)$, our goal is to transform the point cloud $\mathbf{P} = \{p_i \mid i = 1, ..., n\}$ composed of $n$ points into a regular 3D grid representation. Assuming that $\mathbf{P}$ is bounded by $(x_{min}, x_{max})$, $(y_{min}, y_{max})$ and $(z_{min}, z_{max})$ along $x$, $y$ and $z$ axes respectively, we compute a *sparse* occupancy matrix $\mathbf{M}$ using the quantization factors ($\delta_x$, $\delta_y$, $\delta_z$). In this thesis, we utilize dictionary of keys representation to encode $\mathbf{M}$ where the $i^{th}$ point updates $\mathbf{M}$ using:

$$\mathbf{M}\Big[\lfloor\frac{x_i}{\delta_x}\rfloor, \lfloor\frac{y_i}{\delta_y}\rfloor, \lfloor\frac{z_i}{\delta_z}\rfloor\Big] = 1 \qquad i = 1, \ldots, n \tag{5.1}$$

In Equation 5.1, $\Big[\lfloor\frac{x_i}{\delta_x}\rfloor, \lfloor\frac{y_i}{\delta_y}\rfloor, \lfloor\frac{z_i}{\delta_z}\rfloor\Big]$ is analogous to the bin index of $p_i$ after quantization. Dictionary of keys ensures the minimal representation for matrix $\mathbf{M}$ as it only allocates one entry for all points that fall into the same cell. Nevertheless, it is also possible to utilize the coordinate-list representation for this purpose. It should be noted that 99.88% of cells are empty (*i.e.* zero) in the dense representation of $\mathbf{M}$. Thus, our representation significantly reduces the amount of data transferred from the CPU to the GPU or embedded neural network accelerator. The sparse matrix $\mathbf{M}$ is scattered into its dense form on the GPU. Figure 5.1 shows selected slices from the dense occupancy matrix of a partial LiDAR sweep.

A close-up of the generated occupancy cuboid in Figure 5.2 shows that sufficient detail

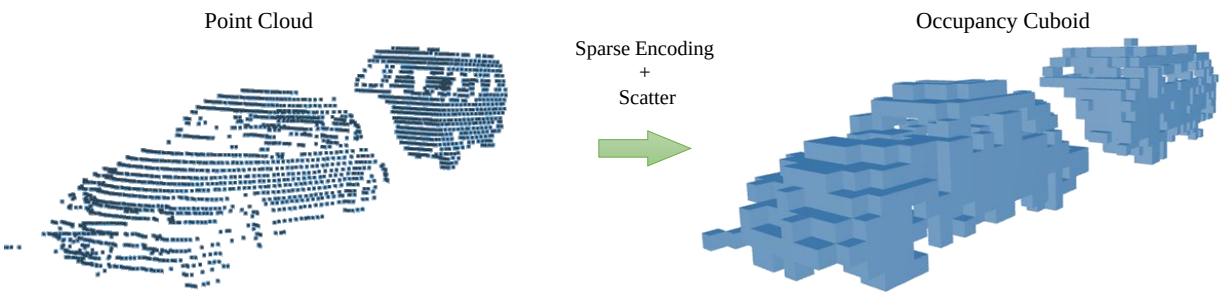



Figure 5.2: Occupancy Cuboid Generation

is preserved for the neural network to process. In addition to that, our representation has a regularizing effect due to its tolerance to noise introduced during the data acquisition stage. In other words, assuming $\pm\zeta cm$ scanning error in the LiDAR point cloud, a point will potentially fall into the same cell as long as $|\zeta| \leq \min\{\delta_x, \delta_y, \delta_z\}$.

## 5.2 Architecture

We base our backbone, illustrated in Figure 5.3, on the widely used architecture of [102, 51] which was originally inspired by the HyperNet meta-architecture [45]. The input to our model is a $3\times m$ sparse occupancy matrix $\mathbf{M}$ that is first transformed into a dense occupancy cuboid of size $H \times W \times C$. It is important to reiterate that the number of points in the point cloud is much larger than the number of positive cells ($n \gg m$) thereby reducing the amount of data transferred. Afterwards, the dense tensor passes through three convolutional blocks with regular 2D convolutions. The first convolution in a block has a stride of 2 to reduce the spatial dimensions while all consecutive convolutions have a stride of 1. We apply nearest-neighbor upsampling followed by a convolution operation to the outputs of Block 2 and Block 3. The feature map from Block 1 is passed through a convolution without upsampling because it has the same spatial dimensions as the target map. Next, we concatenate feature maps from all three blocks. Concatenating feature maps from different levels improves the flow of gradients to intermediate layers and eliminates the need for residual blocks. Moreover, a multi-scale feature map enables the network to exploit both high level and low level features. Finally, $1 \times 1$ convolutions are applied to compute classification and

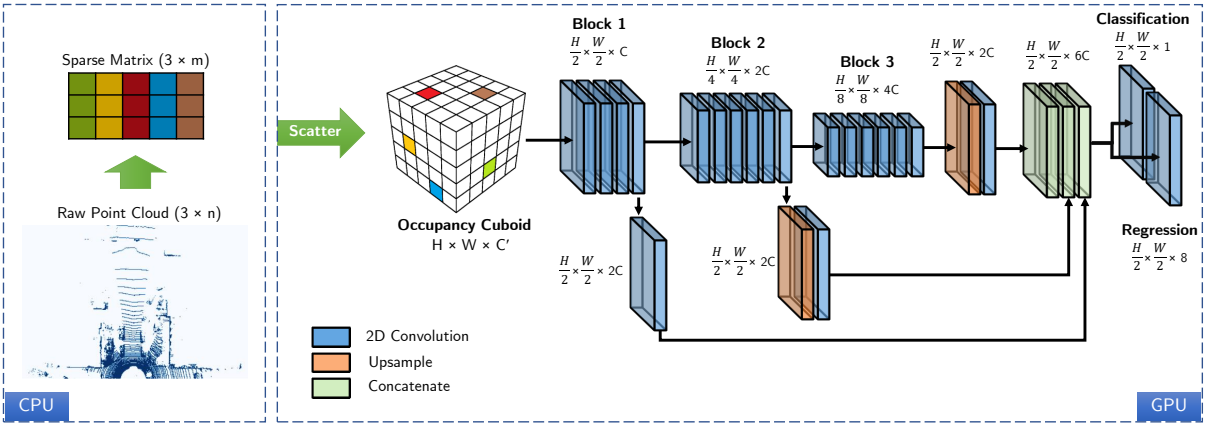



Figure 5.3: Architecture of FA3D

regression outputs. The resulting classification map is activated with a sigmoid to provide the probability distribution of bounding box centers while the regression head estimates the quantities describes in Section 5.3.

We employ a batch normalization layer [41] between each convolution and activation layer to reduce the covariate shift in the input feature maps. However, our empirical findings show that similar performance can be achieved without batch normalization in our network. Regardless, during inference, most efficient frameworks absorb the batch norm computation into the preceding convolution kernel to reduce latency. It is also worth mentioning that we do not use residual blocks [37] as they increase the computational complexity of our model due to the $1 \times 1$ residual connections. Likewise, achieving a comparable performance using feature pyramid networks (FPNs) [58] was not possible with similar computational complexity as our network.

## 5.3 Target Representation

Modern object detectors employ a set of predefined bounding boxes called *anchors* to inject prior knowledge about the dimensions, locations and aspect ratio of bounding boxes. These boxes are placed at regular intervals so as to tile the entire workspace. The use of anchors enforces a near zero mean and unit variance target distribution thereby allowing the network to converge faster.

As illustrated in Figure 5.4, for a ground truth box **G** located at $(x_g, y_g, z_g)$ and having a height $h_g$, width $w_g$ and length $l_g$, we compute a corresponding residual target $\mathrm{t} = (\Delta x, \Delta y, \Delta z, \Delta h, \Delta w, \Delta l)$ relative to anchor **A** using Equation 5.2. Anchors are matched based on their relative offset from the center of a ground truth box. For the same ground truth box defined above, any anchor **A** is positive if its center (with respect to the ground truth coordinate frame) falls within $[x_g - \frac{w_g}{4}, x_g + \frac{w_g}{4}]$ and $[y_g - \frac{h_g}{4}, y_g + \frac{h_g}{4}]$. All remaining anchors that do not satisfy this requirement will become negative.

$$\begin{aligned} \Delta x &= \frac{(x_g - x_a)}{\sqrt{{l_a}^2 + {w_a}^2}} & \Delta h &= \log \frac{h_g}{h_a} \\ \Delta y &= \frac{(y_g - y_a)}{h_a} & \Delta w &= \log \frac{w_g}{w_a} \\ \Delta z &= \frac{(z_g - z_a)}{\sqrt{{l_a}^2 + {w_a}^2}} & \Delta l &= \log \frac{l_g}{l_a} \end{aligned} \tag{5.2}$$

where $x_a, y_a, z_a$ are offsets and $h_a, w_a, l_a$ are dimensions of the anchor.

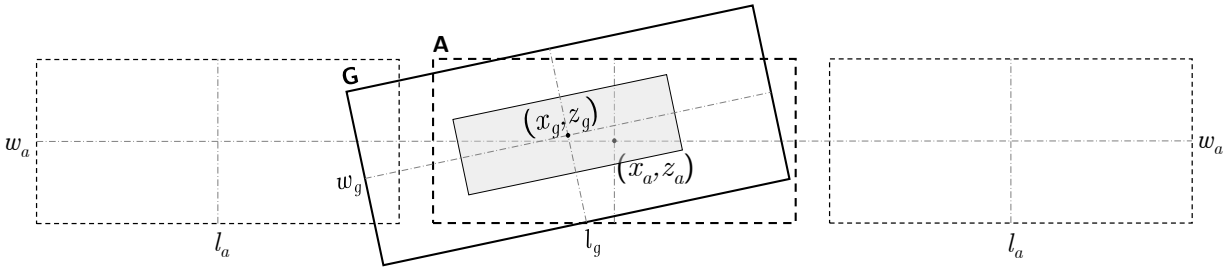



Figure 5.4: Anchor **A** is assigned to detect ground truth **G** because its center is inside the shaded region. Anchors usually overlap each other to maximize detection rate. Overlapping anchors are omitted in this figure for the sake of clarity.

In addition to dimension and location, we must estimate the orientation of each 3D bounding box. In its raw form, the orientation measured in radians is discontinuous [108] (See Figure 5.5). Specifically, for a ground truth box oriented at 0 rads, a network that predicts a value close to $2\pi$ will incur a high loss despite the two values being almost identical. VoxelNet [109] circumvents this by using two anchors facing opposite directions and constraining the offset between 0 and $\pi$. [51, 102] on the other hand use a sine-error loss along with a direction classifier. Other representations such as corner encoding [14]

and vector encoding [48] have been proven to show inferior performance due to redundant information.

The dataset in-use assumes that all objects are bound to the horizontal plane. Hence, the pitch and roll are automatically zero leaving yaw to be the only angle to be regressed. We use a straightforward approach similar to [84] and decompose the yaw orientation vector into its orthogonal basis vectors. For a ground truth box with yaw equal to $\theta_g$, we compute the basis targets $\Delta\theta_x$ and $\Delta\theta_y$ as Equation 5.3. For example, the basis vectors from the previously mentioned discontinuity points (0 and $2\pi$) are $\cos(0) = \cos(2\pi) = 1$ and $\sin(0) = \sin(2\pi) = 0$. As the resultant values are equal, the orientation loss between the two points will be the same and hence eliminate the discontinuity.

$$\Delta\theta_x = \cos(\theta_g - \theta_a) \qquad \Delta\theta_y = \sin(\theta_g - \theta_a) \tag{5.3}$$


Figure 5.5: Discontinuity in direct representation of angles

## 5.4 Loss Functions

Cross entropy is the most common choice for classification tasks. However, it is known to perform poorly for object detection were the negative samples far outnumber positive samples. Models trained with vanilla cross entropy tend to favour the dominant class (background in this case). There are two widely used ways of resolving class imbalance. (1) The number of negative to positive samples ratio is capped to a predefined value, often 3. Additionally, the negative quota is filled with previously misclassified instance, hence

the name *hard-negative mining*. (2) Focal loss [59] introduces additional weights to the standard cross entropy loss to suppress the contribution of easy examples. For training the classification head of FA3D, we use the focal loss specified in Equation 5.4 with $\alpha = 0.75$ and $\gamma = 1.0$.

$$\mathcal{L}_{cls} = -\sum_{i \in pos} \alpha \cdot (1 - \hat{c}_i)^{\gamma} \log(\hat{c}_i) - \sum_{i \in neg} (1 - \alpha) \cdot (\hat{c}_i)^{\gamma} \log(1 - \hat{c}_i) \tag{5.4}$$

where $\hat{c}_i$ is the predicted confidence.

We use smooth L1 loss for the regression head. This loss is computed on positive anchors only as:

$$\mathcal{L}_{reg} = -\sum_{i \in pos} \sum_{m} L_{1(smooth)}(m_i - \hat{m}_i) \tag{5.5}$$

where $m \in (\Delta x, \Delta y, \Delta z, \Delta h, \Delta w, \Delta l, \Delta \theta_x, \Delta \theta_y)$, and $m_i$ and $\hat{m}_i$ are target and predicted quantities respectively.

The total loss (Equation 5.6) of the model is a weighted sum of the classification loss, regression loss and weight decay.

$$\mathcal{L} = \frac{1}{N} (\mathcal{L}_{cls} + \alpha \mathcal{L}_{reg}) + \lambda ||\mathbf{W}||_2^2 \tag{5.6}$$

where $N$ is the number of positive anchors, $\alpha$ is the weight of the regression loss, $\mathbf{W}$ is the parameter vector and $\lambda$ is the regularization constant.

## 5.5 Implementation Details

### 5.5.1 Workspace and Occupancy Cuboid

Similar to [109, 102, 51], we bound the point cloud within a region of $[-40, 40] \times [-1, 3] \times [0, 70]$ m for cars and a region of $[-20, 20] \times [-0.5, 2.5] \times [0, 48]$ m for pedestrians and

cyclists. This covers 98% of labeled ground truth boxes in the KITTI dataset. To generate the binary occupancy cuboid, we set $\delta_x$ and $\delta_z$ to 0.16 and $\delta_y$ to 0.1. Thus, the resulting occupancy cuboid will be of size $H = 500$, $W = 440$ and $C' = 40$.

### 5.5.2 Augmentation

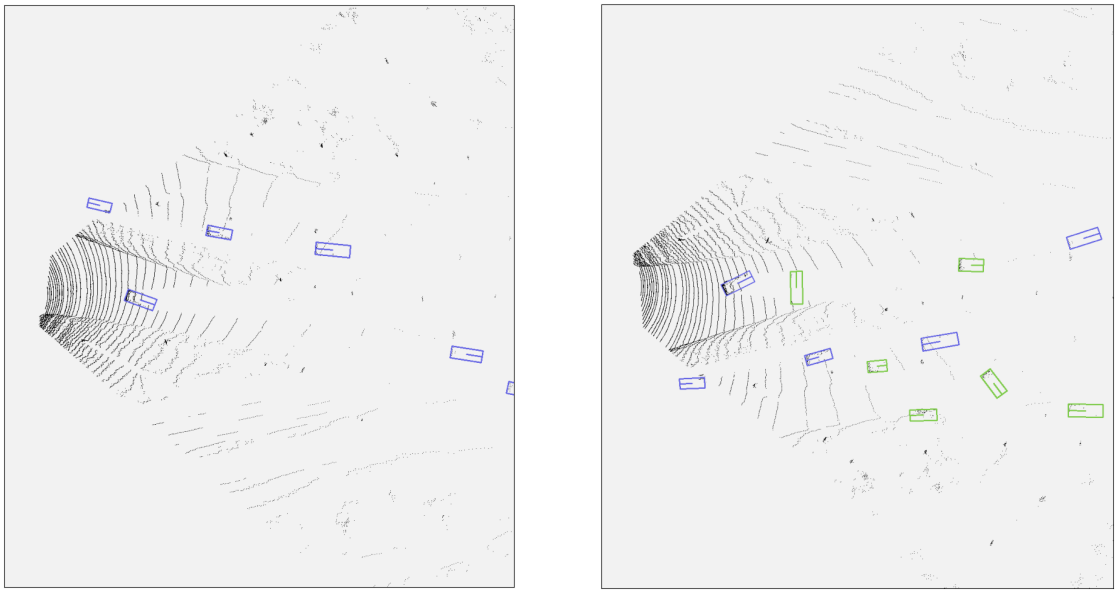
Augmentation is important in reducing overfitting on small datasets such as KITTI. Similar to [102], we create a dictionary of the ground-truth boxes in the training set. During training, we randomly sample five to ten instances for *cars*, eight samples for *pedestrians* and eight samples for *cyclists*, and add them to the point cloud after collision testing. In addition, we estimate the ground plane using RANSAC [9] and adjust the altitude of newly added objects accordingly. Furthermore, ground truth boxes are rotated and translated using values drawn from the uniform distribution $\mathcal{U}(-\frac{\pi}{15}, \frac{\pi}{15})$ and the normal distribution $\mathcal{N}(0, 0.25)$, respectively. The scene is also randomly flipped with a probability of 0.5. In the end, the scene is scaled randomly in the range of $[0.95, 1.05]$, translated by $\mathcal{N}(0, 0.2)$ and rotated by $\mathcal{U}(-\frac{\pi}{4}, \frac{\pi}{4})$. We show an example of an augmented point cloud in Figure 5.6.

### 5.5.3 Network Parameters

The number of kernels for each convolution layer within a block stays the same. The depths for Block 1, Block 2, and Block 3 (Figure 5.3) are $C = 64$, $2C$ and $4C$, respectively. The first block is composed of four convolution layers, and the next two blocks contain six convolution layers each. Moreover, the spatial dimensions of the input feature map to each block are halved by using convolution layer with stride 2. The output from each block is rescaled to $250 \times 200$ using nearest-neighbor interpolation followed by a convolution operation with $2C$ filters.

### 5.5.4 Anchors

We fix the dimensions $(h_a, w_a, l_a)$ of the car, pedestrian and cyclist anchors to $(1.56, 1.6, 3.9)$, $(1.73, 0.6, 0.8)$, $(1.73, 0.6, 1.76)$ respectively. As most objects are laying flat on the ground



(a) Original Point Cloud (b) Augmented Point Cloud

Figure 5.6: Point Cloud Augmentation in bird's eye view: Six cars have been added to the scene and small random perturbations have been applied to all boxes. (Original Boxes - Purple and New Boxes - Green)

plane, we utilize only one anchor along the y-axis. For cars, anchors are centered at $y = -1.0$ m while anchors for remaining classes are placed slightly higher at $y = -0.6$ m. The orientation for all anchors is set to $\theta_a = 0$.

### 5.5.5 Optimization

Our proposed neural network is trained end-to-end using Adam Optimizer [43] for 200 epochs. We set the batch size to 2 and the initial learning rate to $2 \times 10^{-3}$. The learning rate is exponentially decayed by a factor of 0.8 every 15 epochs. The L2 regularization constant $\lambda$ in Equation 5.6 is set $1 \times 10^{-4}$ and the regression weight $\alpha$ equals 2.

### 5.5.6 Post-processing

During inference, we remove all boxes with prediction confidences less than 0.1. Then, we eliminate redundant detections using the fast distance based non-maximum suppression method specified in Section 4.4. We use a distance threshold of 1.5 m, but performance

is consistent for a wide range of values. Computing the distance between two rotated bounding boxes is much faster than computing the IoU while the former is practically adequate to detect an overlap.

### 5.5.7 Framework

We implemented our deep neural network in Python 3 and Tensorflow 1.15 [1]. During training, we optimized some of the pre- and post-processing stages using Numba [50]. Numba compiles the slow interpreted python code into fast machine code. The models were trained on two GTX 1080 Ti GPUs but reported inference speeds were measured on a single RTX 2080 Ti.

## 5.6 Results

### 5.6.1 Comparison with Prior Work

#### Average Precision

We evaluate the performance of our method on detecting cars using the validation set at two different IoU thresholds. As shown in Table 5.1 and the corresponding PR curves in Figure 5.8, our approach performs on par with state-of-the-art methods on BEV and 3D object detection. Moreover, it surpasses PointPillars [51], the fastest published method, on the *Easy* and *Moderate* difficulty levels. Compared to the highest-scoring approaches on the KITTI Leaderboard, our model has a similar AP on *Hard* but slightly lower on the other two levels when evaluated at IoU threshold of 0.5. The speed-accuracy tradeoff of the proposed model and other state of the art detectors is illustrated in Figure 5.7.

Furthermore, we assess our network's performance on vulnerable road users in Table 5.2. Compared to vehicles, pedestrians and cyclists are harder to detect for several reasons. (1) The LiDAR registers fewer points per object due to the smaller surface area. (2) There is a higher magnitude of intraclass variation in dimension due to the deformable nature of

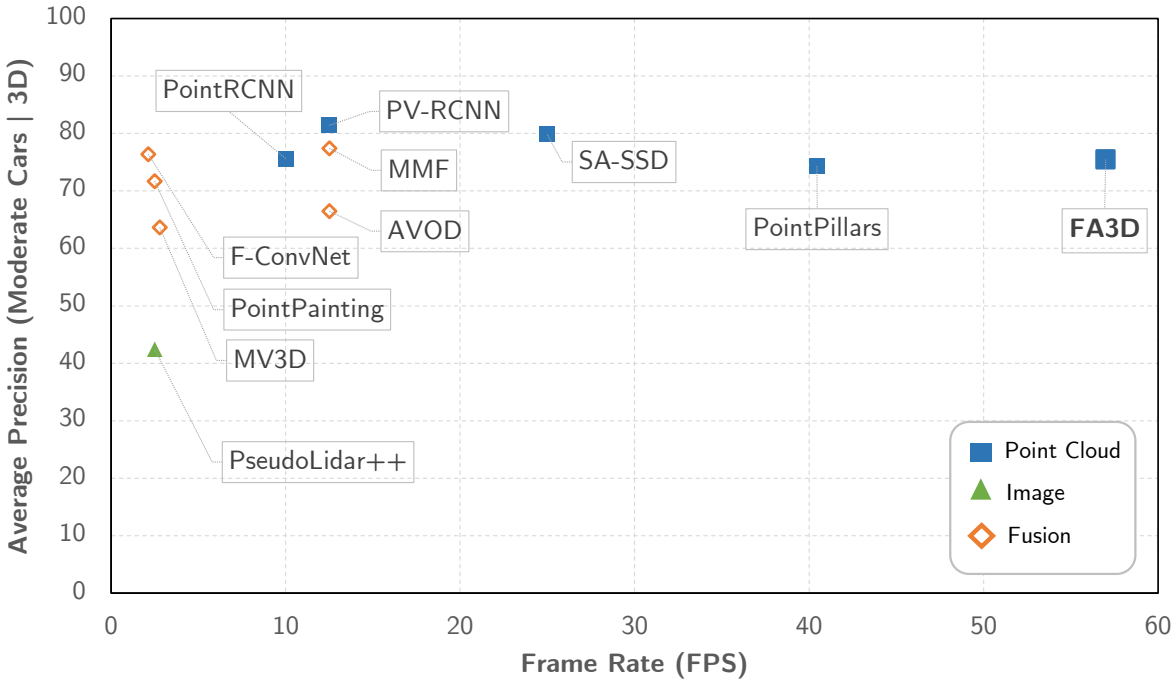



Figure 5.7: Average precision plotted against FPS for FA3D and select 3D object detectors

these classes. Overall, our model detects both classes more accurately than PointPillars but, performs subpar on cyclists compared to the point based methods Part-$A^2$ [83] and PV-RCNN [81]. Note that SA-SSD [35], the current front-runner on the KITTI BEV benchmark, has only reported results on detecting cars. Thus, its detection accuracy on pedestrian and cyclists is not included in our tables.

From a practical standpoint, predicting very tight 3D bounding boxes provides negligible benefit to subsequent tasks in the pipeline such as motion planning. Figure 5.10 illustrates the effect of a slight offset between the ground truth box and the predicted box on IoU considering various scenarios. In all configurations, the IoU drops below 0.7 for an offset of 0.4 meters and below 0.5 for an offset of 0.8 meters. Since our method is designed for fast processing on low-end embedded devices for ADAS applications, and not complex autonomous functions, the 0.4 to 0.8 offset error is still admissible. Thus, evaluating at an IoU threshold of 0.5 is pragmatic for real-world use cases. Furthermore, human-annotated datasets such as KITTI contain noisy labels; and models that report high performance at a high IoU threshold are more likely to overfit to the peculiarities of the annotation pro-

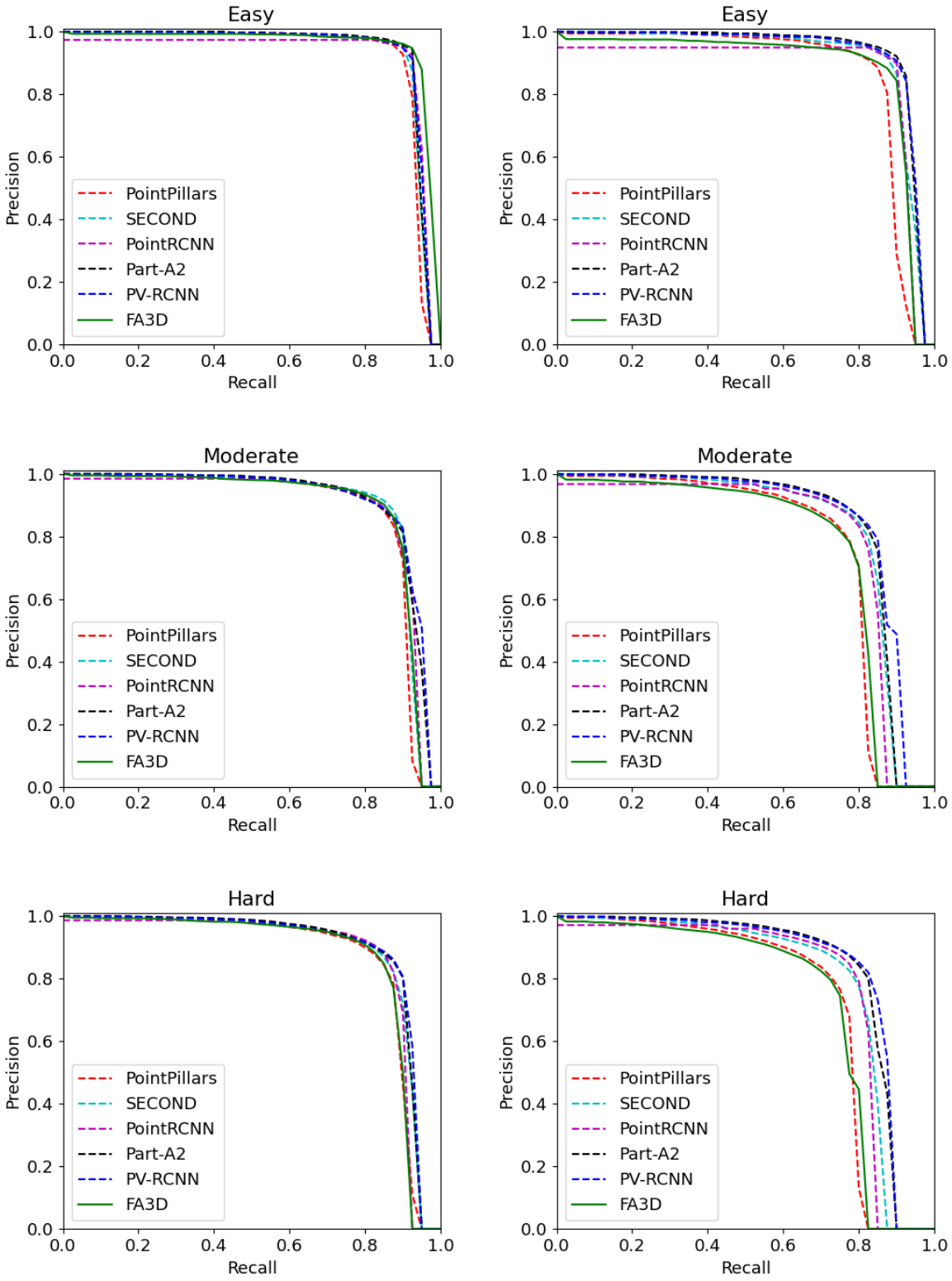
Table 5.1: Comparison with state-of-the-art on cars

(a) IoU Threshold = **0.7**

| | FPS | BEV | | | 3D | | |
|---|---|---|---|---|---|---|---|
| | | **E** | **M** | **H** | **E** | **M** | **H** |
| **PointPillars** [51] | 40 | 91.49 | 87.20 | 85.81 | 86.00 | 75.60 | 72.56 |
| **SECOND** [102] | 20 | 92.41 | 88.55 | 87.64 | 90.54 | 81.60 | 78.59 |
| **PointRCNN** [82] | 10 | 91.39 | 88.28 | 86.31 | 86.65 | 79.30 | 77.26 |
| **Part-A²** [83] | 12.5 | 92.87 | 89.98 | 88.34 | 92.07 | 82.86 | 81.93 |
| **PV-RCNN** [81] | 12.5 | 92.95 | 90.28 | 88.50 | 91.94 | 84.25 | 82.41 |
| **SA-SSD** [35] | 25 | 96.50 | 92.62 | 90.08 | 92.89 | 84.16 | 81.26 |
| **FA3D (Ours)** | 58 | 94.63 | 88.10 | 85.49 | 87.35 | 75.47 | 71.97 |

(b) IoU Threshold = **0.5**

| | BEV | | | 3D | | |
|---|---|---|---|---|---|---|
| | **E** | **M** | **H** | **E** | **M** | **H** |
| **PointPillars** [51] | 94.81 | 93.78 | 93.11 | 94.79 | 93.39 | 92.33 |
| **SECOND** [102] | 95.68 | 94.84 | 94.24 | 95.66 | 94.74 | 94.05 |
| **PointRCNN** [82] | 95.91 | 94.95 | 92.80 | 95.89 | 94.83 | 92.68 |
| **Part-A²** [83] | 97.80 | 93.95 | 93.90 | 97.78 | 93.87 | 93.74 |
| **PV-RCNN** [81] | 98.22 | 94.56 | 94.41 | 98.21 | 94.50 | 94.30 |
| **SA-SSD** [35] | 99.24 | 96.25 | 93.70 | 99.23 | 96.19 | 93.64 |
| **FA3D (Ours)** | 95.94 | 94.31 | 93.65 | 95.91 | 94.07 | 91.71 |




(a) BEV (b) 3D

Figure 5.8: Precision-recall curves of FA3D and other state-of-the-art detectors (Cars)

Table 5.2: Comparison with state-of-the-art on vulnerable road users

(a) **Pedestrians** at IoU Threshold = 0.5

| | BEV | | | 3D | | |
|---|---|---|---|---|---|---|
| | **E** | **M** | **H** | **E** | **M** | **H** |
| **PointPillars** [51] | 71.21 | 65.55 | 60.22 | 63.36 | 57.78 | 52.10 |
| **SECOND** [102] | 60.73 | 56.56 | 52.13 | 55.94 | 51.15 | 46.17 |
| **PointRCNN** [82] | 65.84 | 58.00 | 51.33 | 62.51 | 54.70 | 47.77 |
| **Part-A²** [83] | 70.53 | 64.20 | 59.24 | 66.88 | 59.68 | 54.61 |
| **PV-RCNN** [81] | 65.93 | 58.48 | 54.13 | 62.71 | 54.46 | 49.86 |
| **FA3D (Ours)** | 72.89 | 65.06 | 59.52 | 63.97 | 56.50 | 49.86 |

(b) **Cyclists** at IoU Threshold = 0.5

| | BEV | | | 3D | | |
|---|---|---|---|---|---|---|
| | **E** | **M** | **H** | **E** | **M** | **H** |
| **PointPillars** [51] | 85.51 | 65.22 | 61.10 | 83.72 | 61.57 | 57.57 |
| **SECOND** [102] | 88.03 | 71.16 | 66.89 | 82.96 | 66.72 | 62.78 |
| **PointRCNN** [82] | 89.72 | 70.90 | 67.89 | 86.82 | 68.18 | 63.74 |
| **Part-A²** [83] | 91.91 | 74.60 | 70.61 | 90.23 | 70.04 | 66.91 |
| **PV-RCNN** [81] | 93.46 | 74.54 | 70.11 | 89.10 | 70.38 | 66.02 |
| **FA3D (Ours)** | 90.55 | 66.29 | 62.89 | 82.73 | 60.23 | 56.86 |

cedure. Figure 5.9 illustrates a few examples of incorrect annotations where the predicted bounding boxes are better fits to the enclosed point cloud than the ground truth.

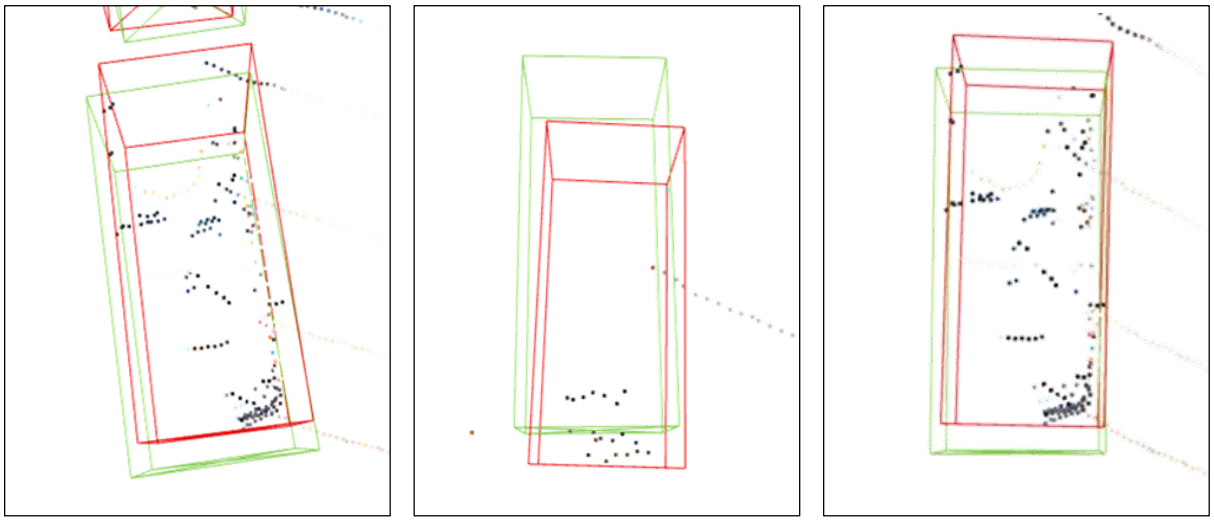


Figure 5.9: Noisy labels in the KITTI Dataset: Green - Ground Truth and Red - Predictions

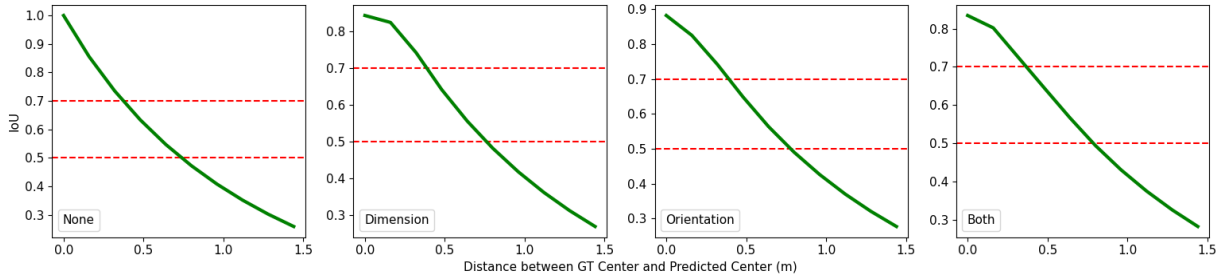



Figure 5.10: Effect of small perturbations on IoU: *None* - ground truth and prediction are identical, *Dim* - slight variation between ground truth and prediction in dimension, *Orientation* - prediction has slight perturbation in orientation, *Both* - both dimension and orientation are different

### Maximum $F_1$ Score

When deploying a model, the confidence threshold is usually set to a value that maximizes the $F_1$ score. Hence, it is more practical to compare the methods based on their maximum $F_1$ score instead of APs for reasons presented in Section 4.3.4. To this end, we compute the maximum $F_1$ score for our method as well as other state-of-the-art methods and compare them in Table 5.3. We observe that the practical performance of highly computational

methods such as SA-SSD [35] is comparable to the proposed network while our method is significantly faster.

Table 5.3: Maximum $F_1$ score for cars

| | **IoU = 0.5** | | | **IoU = 0.7** | | |
|---|---|---|---|---|---|---|
| | **E** | **M** | **H** | **E** | **M** | **H** |
| **PointPillars** [51] | 94.73 | 91.53 | 90.00 | 91.41 | 86.60 | 85.09 |
| **SECOND** [102] | 94.87 | 92.16 | 90.24 | 92.54 | 88.12 | 85.99 |
| **PointRCNN** [82] | 95.91 | 91.27 | 90.70 | 92.52 | 87.00 | 86.66 |
| **Part-A²** [83] | 95.25 | 90.12 | 90.18 | 93.40 | 86.71 | 86.54 |
| **PV-RCNN** [81] | 95.56 | 91.07 | 90.64 | 92.33 | 86.93 | 86.86 |
| **SA-SSD** [35] | 97.01 | 93.02 | 91.88 | 95.58 | 89.74 | 88.45 |
| **FA3D (Ours)** | 95.86 | 91.47 | 89.44 | 93.61 | 87.54 | 85.38 |

#### Quantization Factor

Quantization factor of the occupancy grid plays an important role in overall latency and accuracy of the detector. We plot the the average precision against quantization factor at 4 cm intervals in Figure 5.11. Surprisingly, our findings suggest that smaller cell sizes do not always yield higher performance. For example, the network trained with 16 cm bins has a higher AP than the network with 8 and 12 cm bins. Nonetheless, the general trend line indicates an inverse relationship where performance drops as the quantization factor increases.

### 5.6.2 Runtime Analysis on Off-the-shelf Hardware

As of writing, FA3D reported the shortest latency on the KITTI benchmark. Prior to that, the position was held by PointPillars [51]. Therefore, it is necessary to compare the latency of both models and identify the key areas of improvement. For this reason, we run and time the models on two commercially available computers. The first machine is

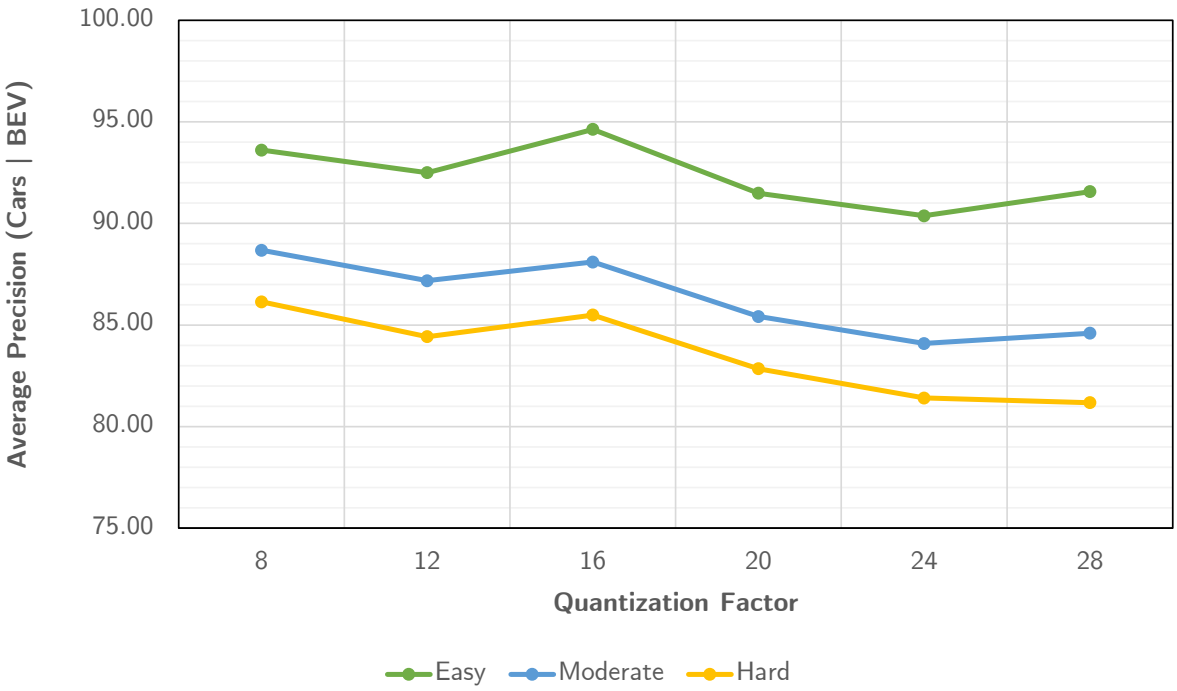



Figure 5.11: The effect of quantization factor on performance

a high-end workstation equipped with a Core-i7 CPU and a RTX 2080 Ti graphics card while the second is a Jetson AGX Xavier — a single board embedded computer with an 8-core ARM processor and 512-core Volta GPU. The details of both configurations can be found in Table 5.4. We used the official PyTorch implementation [68] of PointPillars [51] for all timing analysis. It is worth mentioning that we have not optimized our model using TensorRT or quantized the weights to make a fair comparison with PointPillars [51].

Table 5.4: Specifications of the benchmarking hardware

| | **Workstation** | **Jetson AGX Xavier** |
|---|---|---|
| **CPU** | Core i7 | Octa-Core Carmel ARM v8.2 |
| **CPU Clk. Speed** | 3.8 GHz | 2.26 GHz |
| **RAM** | 64 GB | 32 GB |
| **GPU** | GTX 2080 Ti | 512-Core Volta |
| **GPU Architecture** | Turing | Volta |
| **Max. TOPs** | 430 | 32 |

The inference procedure is broken down into encoding, forward pass, and post-processing. Table 5.5 shows the latency of each stage averaged over 100 non-consecutive frames. Additionally, inference time on the Jetson is reported for multiple power models. Our approach is faster than PointPillars in all three stages. As expected, the highest reduction in latency was observed in the input encoding procedure. Our encoder requires 3.09 ms less time (on MAX-N) making it almost six times faster than pillar encoding. Furthermore, the forward pass takes 14.34 ms on the workstation and 125.26 ms on the embedded board making it marginally faster than PointPillars. Overall, our model is slightly more accurate than PointPillars and processes 7.27 point clouds per second on a Jetson AGX Xavier and 57.83 point clouds per second on a high-end workstation, which is 18% and 43% faster compared to PointPillars, respectively.

Table 5.5: Comparing the latency of our proposed method with PointPillars on a high-end workstation and Jetson Xavier using different power settings. (All measurements, except FPS, are in milliseconds (ms))

| | | **Jetson AGX Xavier** | | | **Workstation** |
|---|---|---|---|---|---|
| | | **15W** | **30W-All** | **Max-N** | |
| **PointPillars** [51] | Encoder | 6.45 | 5.44 | 3.71 | 4.08 |
| | Network | 248.41 | 195.06 | 139.73 | 15.69 |
| | Post Proc. | 29.37 | 27.60 | 19.05 | 5.02 |
| | FPS | 3.08 | 4.38 | 6.15 | 40.45 |
| **FA3D (Ours)** | Encoder | 1.31 | 1.32 | 0.62 | 0.48 |
| | Network | 242.96 | 184.26 | 125.26 | 14.34 |
| | Post Proc. | 15.38 | 15.35 | 11.60 | 2.47 |
| | FPS | 3.85 | 4.97 | 7.27 | 57.83 |

The efficient input encoding algorithm presented in Section 5.1 is one of the main contributions of this thesis. Thus, it would be interesting to dissect it further and understand the source of the gain. To this end, we skip the sparse representation and voxelize the point cloud into a **dense occupancy grid**. This change has no impact on the detection accuracy but is expected to affect the speed because the dense matrix takes longer to transfer to

Table 5.6: Latency and Frame Rate of Binary Occupancy Cuboids and Pillar Encoding

| | **Latency** | **Frame Rate** |
|---|---|---|
| **Pillar Encoding** [51] | 4.08 | 40.45 |
| **Binary Occupancy Cuboid (Dense)** | 1.95 | 53.30 |
| **Binary Occupancy Cuboid (Sparse)** | 0.48 | 57.83 |

the GPU. Our findings, reported in Table 5.6, show that binary occupancy cuboids, even in dense form, are more than 2 folds faster than the PointNet based Pillar Encoding used in PointPillars [51].

### 5.6.3 Profiling on DesignWare EV6x Vision Processors

In the case of ADAS and AV, time is of the essence. The fact that decisions are made in a fraction of a second inhibits us from relying on cloud computing infrastructure. Moreover, transmitting data over a network is expensive, exposes the driver to potential security and privacy breaches and is highly dependent on the quality of the reception. Consequently, vehicles carry a computer onboard to process the data. GPUs can be used for initial research and development; but are too bulky and power hungry to use in production. Alternatively, DSPs and FPGAs can augment embedded CPUs to run matrix operations. However, both components offer lower throughput compared to GPUs because the former has fewer cores and the latter runs at a lower clock speed. These inference hardware are compared in Figure 5.12.

The DesignWare EV6x Vision Processors from Synopsys is an efficient heterogeneous processing unit that is capable of running several computer vision algorithms in real time. Despite its affordable price and low power consumption, it can execute up to 4.5 trillions MACs per second. As illustrated in Figure 5.13, it integrates one or more instances of:

- **32-bit scalar RISC processor** — runs the main program and orchestrates other modules

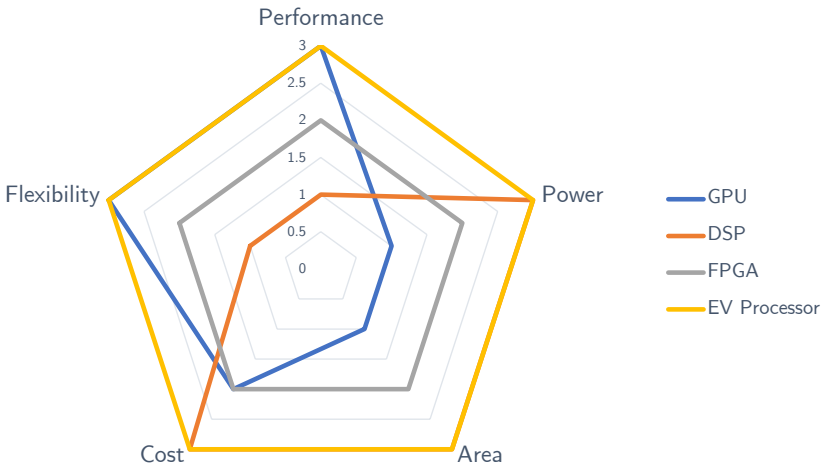



Figure 5.12: Comparison of inference hardware on a 1 to 3 scale (higher is better)

- **512-bit vector digital signal processor (DSP)** — computes simple vector arithmetic in a parallel manner
- **CNN Engine** — does most of the heavy lifting by executing the convolution operations

The RISC processor and the vector DSP are packaged together and referred to as a *core*. Similarly, a single instance of the CNN engine is called a *slice*. Cores and slices can be combined in a number of ways to suit the client's needs. Nevertheless, three configurations [1] are prevalent. On the lowest end, the **EV61+DNN880** features a single core coupled with a single slice. The **EV62+DNN1760** and **EV62+DNN3520** both have a dual-core CPU. However, the first one is fitted with two slices while the latter has four.

We ported and tested FA3D on three of the previously mentioned EV processor configurations. Minor modifications were made to the architecture for better mapping between the model and the hardware. (1) As shown in Figure 5.13, the CPU and CNN Engine have direct access to a shared memory cluster. Unlike a GPU, this dictates that data is not transferred to the CNN accelerator via the CPU. Therefore, the sparse occupancy representation is bypassed and the dense form is generated from the very beginning. (2) The nearest neighbour upsampling layer is replaced with a transpose convolution layer so

[1] **EV64+DNN3520**, which boasts a quad-core CPU, is also available but not investigated in this thesis.

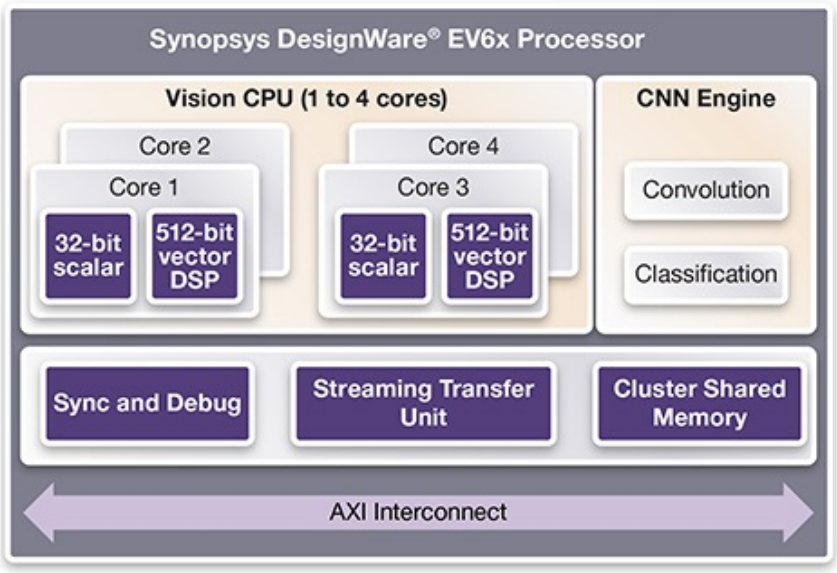



Figure 5.13: DesignWare EV6x Vision Processor Architecture [91]

that the upsampling stage runs on the CNN engine instead of the vector DSP.

As described earlier, the proposed deep neural network has five blocks: three *feature extraction blocks*, an *upsampling and merging block* and a *prediction head*. The first step was to measure the latency of each block on the fast performance model. The fast performance model (FPM) is a numerical model for estimating the latency and other important quantities without requiring an RTL description. It is intended for rapid design iteration prior to running on a physical device. After verification, we profiled the model on an FPGA based prototyping platform called the High-Performance ASIC Prototyping System (HAPS-80). As shown in Figure 5.14, estimates from the FPM are slightly but proportionally lower than the HAPS-80. This phenomenon has been included in the documentation and is expected behaviour. The FPM is ideal for early architectural exploration, but the HAPS-80 is better suited for generating final timing reports. Hence, we exclude FPM estimates, and report only on the HAPS-80 for all subsequent sections.

The latency of the full inference pipeline (refer to Section 5.6.2 for breakdown details) is reported in Table 5.7. FA3D is able to run in near real-time on the first configuration and in real-time on the remaining two. Similar to timing experiments conducted in the previous section, the CNN accounts for most of the processing time. The throughput of

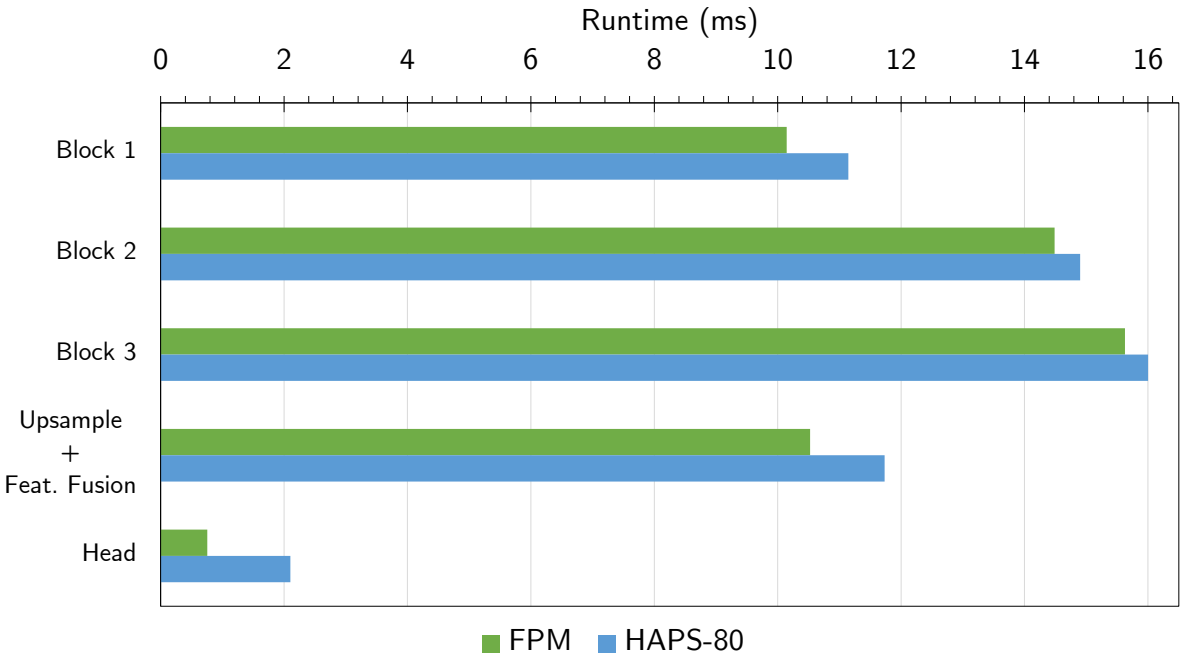



Figure 5.14: Block level comparison of HAPS-80 and FPM on EV61 + DNN880

EV62+DNN3520 was comparable to a high-end workstation while consuming less power. Moreover, we show the efficiency of each block in terms of MAC utilization in Figure 5.15. The feature extractors are the most efficient while the head is the least efficient. Overall, a MAC utilization above 50% can be signed off for deployment.

Table 5.7: Latency on DesignWare EV6x Vision Processors. All measurements, except FPS, are in milliseconds (ms).

| | **EV61+DNN880** | **EV62+DNN1760** | **EV62+DNN3520** |
|---|---|---|---|
| **Encoder** | 0.30 | 0.32 | 0.30 |
| **Network** | 53.58 | 28.17 | 17.59 |
| **Post Proc.** | 0.93 | 0.76 | 0.73 |
| **Total** | 54.81 | 29.25 | 18.62 |
| **FPS** | 18.2 | 34.2 | 53.7 |

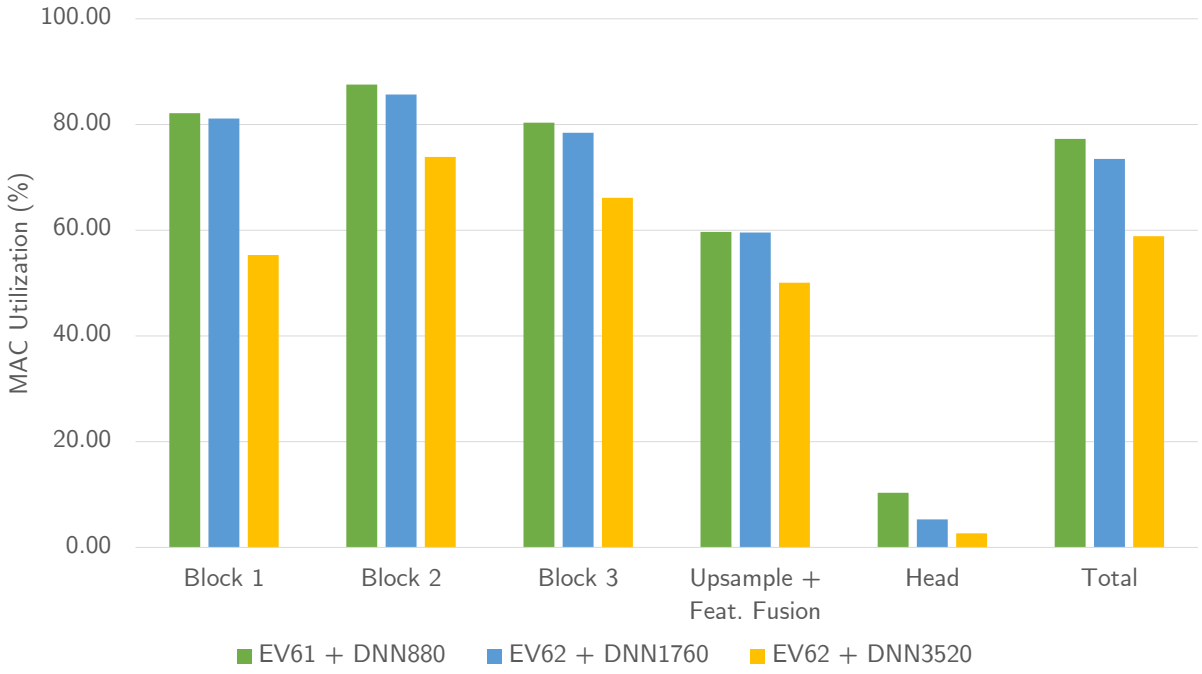



Figure 5.15: Block level MAC utilization on three EV Configurations

### FA3D Lites

Even though the model runs in real-time on the high-end and mid-range EV configurations, EV61+DNN880 was able to process 18 frames/sec only. Thus, we investigated the accuracy-latency trade-off by creating additional models, called FA3D Lites, with reduced kernel count $C$. The original model has $C = 64$ filters in the first block, and 2C and 4C in Blocks 2 and 3, respectively. Figure 5.16 shows a logarithmic relation between the two quantities. The AP at IoU=0.5 drops by 0.25 points for a three folds increase in speed and 5.2 points for 20x speedup. The accuracy deteriorates at a much faster pace when evaluated at IoU=0.7.

### Bandwidth Statistics

Bandwidth, in the context of the EV processor, conveys the maximum rate of data transfer between the CNN Engine and the shared memory cluster. Layer weights and intermediate feature maps are continuously moved between the two modules. Bandwidth is an important, but often overlooked, constraint in neural network accelerator hardwares. As

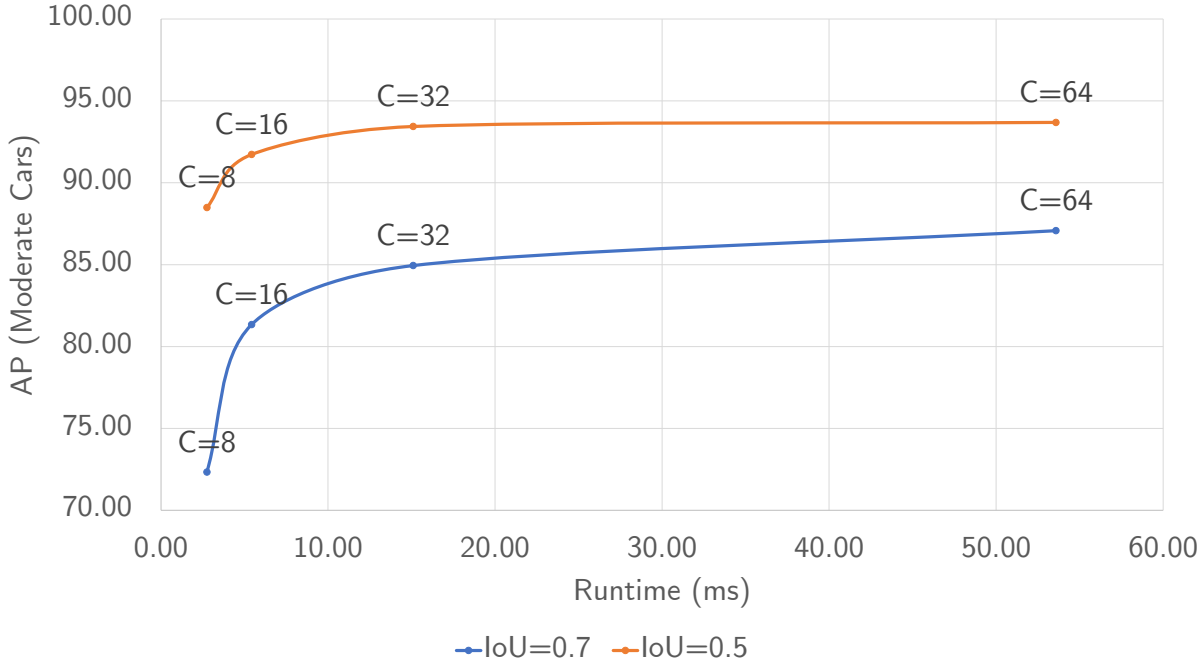



Figure 5.16: Runtime of FA3D Lites on EV61 + DNN880

Table 5.8: Bandwidth statistics (in MB/Frame) on three EV configurations

| | **EV61+DNN880** | | **EV62+DNN1760** | | **EV62+DNN3520** | |
|---|---|---|---|---|---|---|
| | **Read** | **Write** | **Read** | **Write** | **Read** | **Write** |
| **VGG16** [86] | 247.45 | 9.29 | 235.84 | 9.29 | 235.57 | 9.29 |
| **ResNet-101** [37] | 121.33 | 23.89 | 108.17 | 24.42 | 107.89 | 23.93 |
| **YOLO-V3** [76] | 443.50 | 188.11 | 388.31 | 188.11 | 457.20 | 188.19 |
| **SSD** [61] | 126.92 | 22.79 | 94.92 | 22.78 | 105.91 | 27.17 |
| **Faster RCNN** [29] | 1137.58 | 160.36 | 1040.18 | 160.35 | - | - |
| **FA3D (Ours)** | 147.70 | 113.54 | 138.35 | 113.53 | 139.33 | 113.55 |

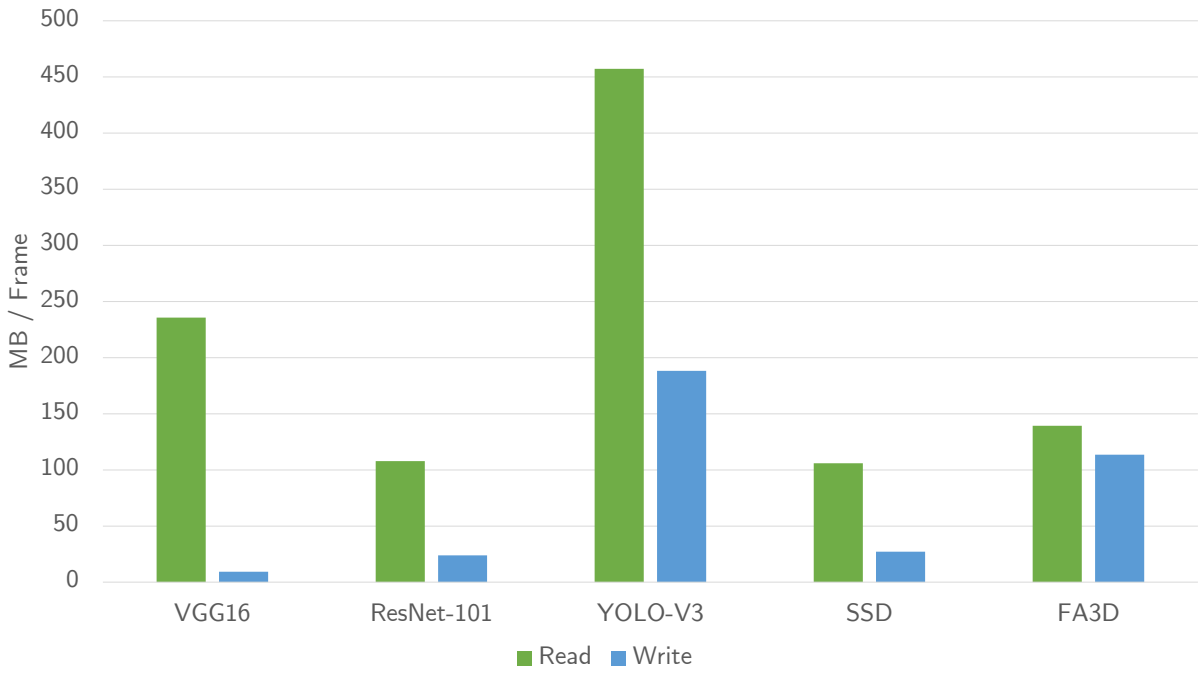



Figure 5.17: Bandwidth statistics on EV62 + DNN3520

reported in Table 5.8, FA3D writes around 114 MB to the shared memory and reads about 140 MB for every frame. Using the latency information provided in Table 5.7, we estimate the bandwidth per frame for the EV62+DNN3520 configuration to be 285 MB/frame. As the sum of the read and write speeds is below the threshold, we confirm that our system is not bandwidth limited.

Optimal mapping of other 3D object detectors to the EV processor is an arduous endeavor. For this reason, we compare our model with other well known image classifiers and 2D object detectors that were previously mapped by the manufacturer. The results are presented in Figure 5.17. Our model writes more data to memory than all the classifiers and SSD [61] but less than YOLO-V3 [6]. Furthermore, the amount of data read per frame by FA3D is close to SSD [61] but one-thirds of YOLO-V3 [6].

### 5.6.4 Error Analysis

In this section, we analyze the contribution of the classification and regression heads towards the overall error of the model. We conduct several sub-tests to narrow down the

sources of error in FA3D.

We start by probing the classification head. We alter its results to asses the impact of FPs and FNs on overall AP. We add all missing detections (FNs) by iterating over all ground truth boxes and appending the ones with IoU less than 0.3 for pedestrians and 0.5 for cars. Similarly, we filter all incorrect detections (FPs) by iterating through all predictions and removing boxes with IoU less than previously mentioned thresholds. It is worth mentioning that the threshold has been lowered to accommodate for boxes that are "close enough" and can be easily corrected by the regression head. The results on both cars and pedestrians are presented in Table 5.9. Compared to removing FPs, adding FNs had a smaller impact on performance. This shows that the model has a high recall. It is even more pronounced for pedestrians, whereby removing FPs resulted in an 11% increase in AP.

Table 5.9: The effect of classification head on average precision

(a) Cars at IoU Threshold = 0.7

| | **BEV** | | | **3D** | | |
|---|---|---|---|---|---|---|
| | **E** | **M** | **H** | **E** | **M** | **H** |
| **None** | 94.63 | 88.10 | 85.49 | 87.35 | 75.47 | 71.97 |
| **Add FNs** | 94.63 | 89.65 | 87.92 | 87.35 | 77.09 | 73.18 |
| **Remove FPs** | 96.22 | 90.61 | 88.17 | 88.20 | 77.09 | 74.46 |

(b) Pedestrians at IoU Threshold = 0.5

| | **BEV** | | | **3D** | | |
|---|---|---|---|---|---|---|
| | **E** | **M** | **H** | **E** | **M** | **H** |
| **None** | 72.89 | 65.06 | 59.52 | 63.97 | 56.50 | 49.86 |
| **Add FNs** | 73.54 | 67.44 | 65.62 | 64.52 | 58.44 | 55.72 |
| **Remove FPs** | 82.49 | 76.00 | 71.37 | 71.15 | 65.03 | 58.58 |

Ideally, for a perfect regression head, the APs would have been close to 100% after replacing the classification head with the ground-truth labels. However, the findings suggest

that even with a precise classification, the regression head is still unable to localize the bounding boxes accurately. Therefore, it is safe for one to assume that the regression head also plays a significant role in the performance of the model. To study this more closely, we selectively replace each output of the regression head with ground truth values. The results are detailed in Table 5.10. For instance, the fourth row in this table states that we replace the predicted $x$ and $z$ values with the ground-truth values and use them along with predicted $y$, $h$, $w$, $l$ and $\theta$ to localize bounding boxes.

Table 5.10: The effect of regression head on average precision (Cars).

| | BEV | | | 3D | | |
|---|---|---|---|---|---|---|
| **Replaced Qty.** | **E** | **M** | **H** | **E** | **M** | **H** |
| **None** | 94.60 | 87.90 | 85.34 | 87.33 | 75.37 | 70.66 |
| **x** | 95.18 | 91.00 | 89.83 | 89.47 | 76.87 | 72.82 |
| **z** | 95.16 | 90.55 | 88.32 | 89.37 | 75.86 | 72.76 |
| **xz** | 94.57 | 92.18 | 90.45 | 92.66 | 83.89 | 81.25 |
| **w** | 94.76 | 88.10 | 85.55 | 89.63 | 76.04 | 73.01 |
| **l** | 94.55 | 88.42 | 87.20 | 90.03 | 78.14 | 73.52 |
| **wl** | 94.75 | 88.56 | 87.53 | 90.28 | 78.63 | 75.52 |
| **y** | 96.68 | 90.18 | 87.50 | 91.13 | 81.69 | 78.86 |
| **h** | 94.60 | 87.99 | 85.39 | 87.75 | 74.62 | 71.18 |
| **yh** | 96.71 | 90.19 | 87.50 | 95.56 | 89.25 | 86.58 |
| **xzwl** | 94.80 | 93.31 | 92.66 | 94.66 | 90.70 | 89.55 |
| **xyz** | 94.59 | 92.47 | 90.50 | 93.49 | 89.17 | 86.92 |
| **hwl** | 97.01 | 90.95 | 88.22 | 89.64 | 78.15 | 75.04 |
| $\theta$ | 94.48 | 87.88 | 85.35 | 87.32 | 75.60 | 71.26 |
| **xyzhwl** | 97.23 | 95.87 | 94.87 | 95.76 | 94.79 | 93.87 |
| **xyzhwl**$\theta$ | 97.23 | 95.92 | 95.05 | 95.77 | 94.84 | 94.04 |

As it turns out, predicting $x$ (lateral axis) more accurately will improve the results dramatically. Likewise, predicting $z$ values (longitudinal axis) precisely will have a significant

impact on the accuracy. The third important regression output is $y$ (vertical axis) values as they improve the results on both BEV and 3D metrics. In contrast, replacing $w$, $h$, $l$ and $\theta$ with ground truth values has less impact on the detection accuracy. The results suggest that our network has already generalized well on these four quantities.

In sum, the KITTI dataset is small and our proposed method overfits on the dataset (even using augmentation). However, most of the regression values generalize well and the main source of error boils down to regressing $x$, $y$, and $z$ values. We argue that using a better augmentation or larger dataset will potentially improve the generalization of these three quantities.

## 5.7 Conclusion

In this chapter, we proposed a fast and accurate 3D object detector that is 18% and 43% faster than the fastest published method in this area on a Jetson Xavier and a high-end workstation respectively. Our network utilizes a computationally efficient, yet expressive input representation that reduces the time required to encode the point cloud and copy it to a GPU. One of the main advantages of our method is that we do not gain time improvements by designing a smaller or more biased network. Quite the contrary, our model is complex and highly flexible. In addition to the detailed accuracy and runtime experiments on three different kinds of neural network accelerators, we also analyzed the predictions and identified quantities with the highest impact on detection error.

# Chapter 6

# Refine 3D Bounding Boxes through Late Vector Fusion

Pioneering datasets such as KITTI [26] have opened new prospects for researchers to exploit multimodal data for improving the accuracy of 3D object detectors. Previously in Chapter 5, we found that both the classification and the regression heads had comparable contribution towards the overall error of FA3D. This chapter presents a late fusion scheme that complements the classification head and improves its precision. We begin by analysing detections from the previous model and identify inspirations for our work in Section 6.1. The proposed architecture is described in Section 6.2 and findings are presented in Section 6.3. At last, we extend our framework to other state-of-the-art 3D object detectors in Section 6.4.

## 6.1 Motivation

We believe that color images are better suited for classification and not regression for three reasons. (1) The rich color and texture information inscribed in images is more valuable for discerning between objects. (2) Previous works [96, 105] have shown that inferring depth from images (*i.e.* 2D projections) is prone to large margins of error. (3) As a result of perspective projection and quantization in the image, small translation errors in image

space are amplified in physical space. For example, in Figure 6.1, an offset of 3 pixels in the image equates to approximately 30 cm in the real world. For these reasons, we direct our fusion efforts in improving the classification head and postpone research on the regression head for future work.

To study the classification errors more closely, we plot the number of TPs, FPs and FNs as a function of distance for cars and pedestrians (See Figure 6.2). One can observe that the number of FPs far exceeds FNs especially for pedestrians. Too many false detections lead to frequent braking, which in turn causes a jerky and unpleasant driving experience. On the extreme end, it can also result in a rear-end collision. On the other hand, occasional FNs in ADAS are neither bothersome nor fatal because the system expects the human driver to remain engaged and actively identify them. In this chapter, we propose a late fusion scheme that uses segmentation maps generated from an image to complement the classification head of FA3D and reduce the number of false detections. Overall, our goal is to improve the performance of our previously proposed 3D object detector using RGB images.

Figure 6.1: Translating a 2D bounding box by 3 pixels in image space corresponds to approximately 30 cm in physical space

## 6.2 Architecture

The pipeline of our proposed fusion framework is illustrated in Figure 6.3. It consists of two concurrent data processing routes (image stream and point cloud stream) and a simple

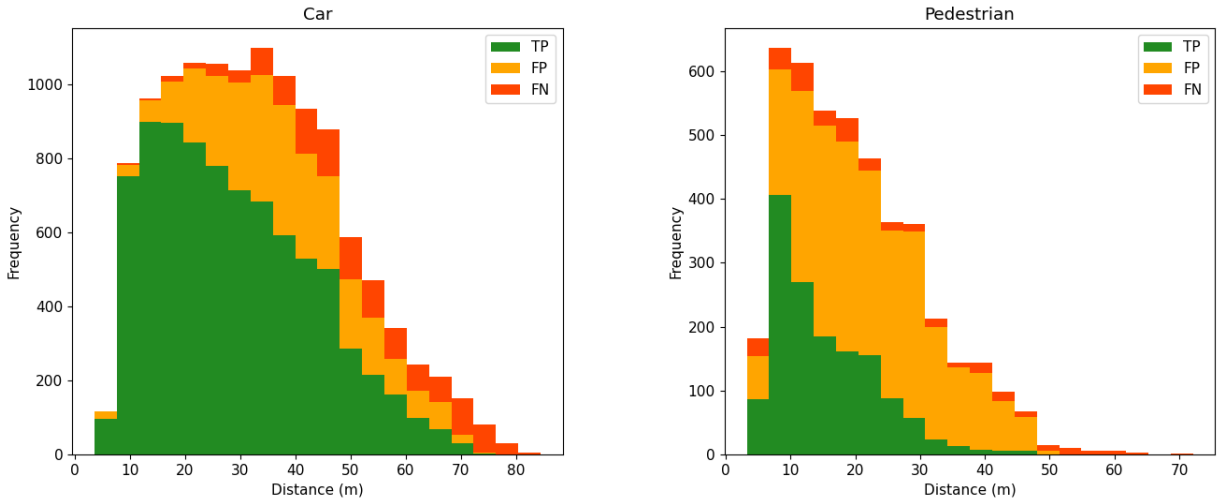



Figure 6.2: Stacked bar plot of the classification statistics (TPs, FPs and FNs) as a function of distance for cars and pedestrians (Confidence Threshold = 0.5)

but effective fusion module. Compared to sequential fusion schemes such as PointPainting [95] and F-PointNets [70, 97], our parallel setup introduces minimal runtime overhead to the system. The overall time-to-completion of two concurrent pipelines is not computed as the sum of the two streams but rather bounded by the slowest stream. Particularly, in our implementation, the image stream (with a latency of 34 ms) is slower than the point cloud stream (with a latency of 17.2 ms) and hence caps overall throughput at 28 FPS. If needed, the image stream can be replaced with a more recent real-time image segmentation network such as DDRNet [38].

### 6.2.1 Image Stream

The goal of the image stream is to convert an input image of $H' \times W' \times 3$ into a segmentation vector of $W' \times K$, where $K$ is the number of classes and $H'$ and $W'$ are the image's height and width respectively. For our experiments, we use KITTI's default image dimensions of $H = 375$ and $W = 1242$ and set $K$ to 2 (car and pedestrian). It takes two simple steps to compute the segmentation vector. For the first step, a pretrained semantic segmentation network is used to convert the image into a pixel-wise semantic segmentation map with a size of $H' \times W' \times K$. Afterwards, we extract the maximum value along the y-axis of the segmentation map to generate a $W' \times K$ tensor. Intuitively, this tensor can be seen as

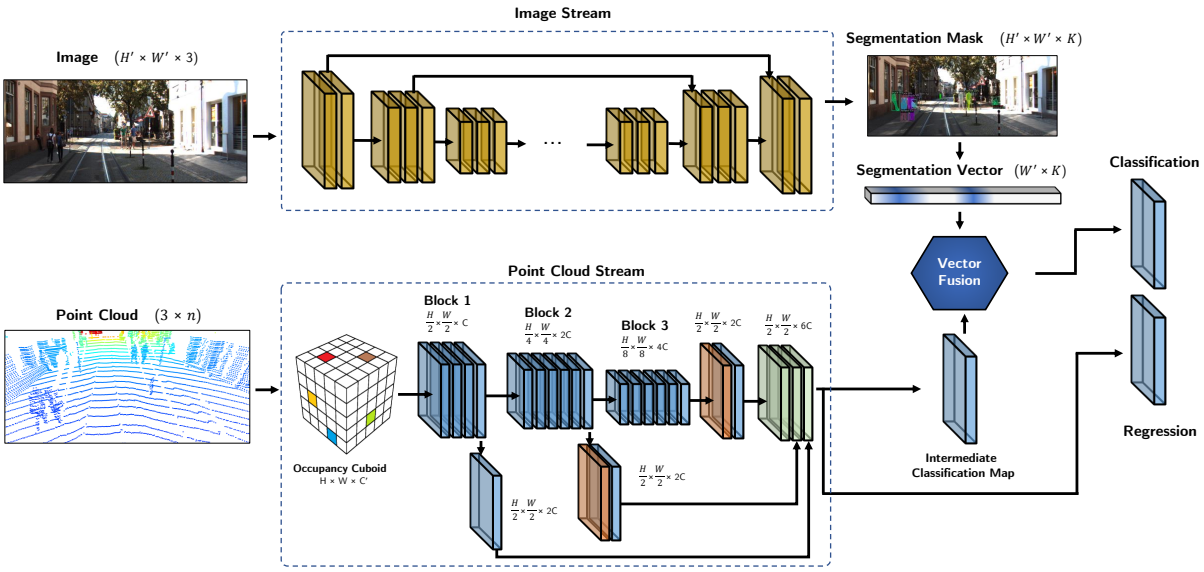



Figure 6.3: Block Diagram of VectorFusion

a group of $K$ vectors with length $W$ where each vector indicates the likelihood of finding an object along the x-axis. The main motive behind compressing the segmentation map along the y-axis is to suppress the error introduced when regressing altitude $y$ and height $h$ by the proposal network.

A robust segmentation mask is vital for our framework. Consequently, we use a mature publicly available implementation [2] of Mask R-CNN [36] that is based on a ResNet-101 backbone [37] and an FPN [58], and trained on the MS COCO dataset [60]. Even though we use a segmentation network, similar results can be achieved through a 2D object detector. However, there are several benefits in using a semantic segmentation network. (1) Segmentation networks are easier to design and implement due to the one-to-one correspondence between input and output. Moreover, they do not involve prior assumptions (*i.e.* anchors) about the dimensions of objects. (2) The losses used in optimizing an object detection network create a dilemma between the classification head which is translation-invariant and the regression head which is translation-variant [17]. (3) Semantic segmentation maps are used for additional perception tasks such as estimating drivable space and detecting traffic lights and signs.

### 6.2.2 Point Cloud Stream

We use FA3D which was elaborated in Chapter 5 for processing the point cloud. For the sake of convenience, we summarize the chapter as follows. First, the point cloud is voxelized into an occupancy cuboid and passed through a series of convolution blocks. Three intermediate hierarchical feature maps are resized and concatenated to create a multi-scale representation of the input data. Finally, 1 x 1 convolutions are used for generating two sibling heads: classification and regression. The classification map is passed to the fusion module for further processing. As presented at the beginning of the chapter, refining 3D pose and dimensions from RGB images is not a trivial task. Thus, the regression map from the point cloud stream is used directly to generate 3D bounding boxes.

### 6.2.3 Vector Fusion

This module consolidates the BEV classification map from the point cloud stream with the segmentation vector from the image stream. This is achieved by mapping points in the classification map to the segmentation vector and then purging FPs.

At first, the classification map is thresholded by $\beta = 0.1$. This hyperparameter can be adjusted to control the number of positive point proposals from the point cloud stream based on the availability of compute power. Fixing the number of proposals to a constant value by picking the top-k points is another approach and ensures similar runtime in the fusion module regardless of the input. After thresholding, for each positive point proposal $p$ located at $(z, x)$ in BEV, we compute its projection $\tilde{x}$ on the segmentation vector as:

$$\tilde{x} = \frac{f}{f+z} \cdot x + \frac{W'}{2} \tag{6.1}$$

where $f$ is the focal length of the camera and $W'$ is the width of the image. Refer to Figure 6.4 for a visualization of the projection step.

Similarly, we define a window $\Delta$ in Equation 6.2 that captures the average size of each object class in vector space. It is important to note that the window depends on the

location of the proposal from the sensor and magnitude of the longest dimension $l_a$ for the class in consideration. We use the same values used for anchors in Chapter 5 and fix $l_a$ to 3.9 for cars and 0.8 for pedestrians.

$$\Delta = \frac{f}{f+z} \cdot l_a \tag{6.2}$$

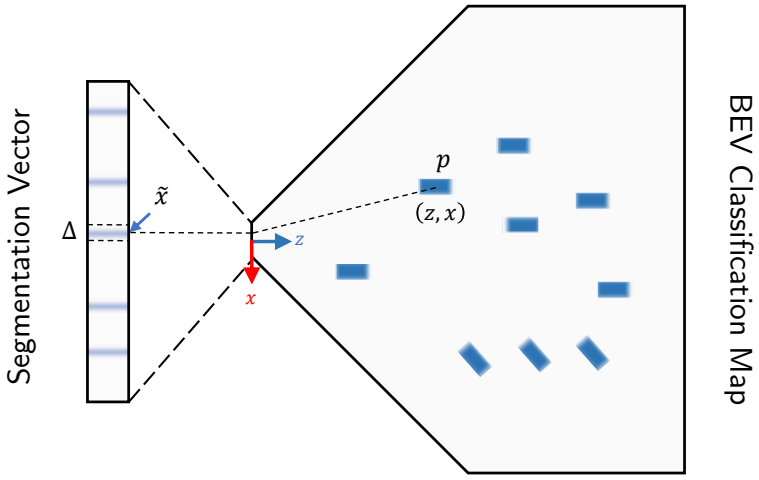



Figure 6.4: Projecting positive point proposal from BEV to Segmentation Vector

In the end, these two values can be used to compute the positive density $\rho$ of each point proposal. The positive density, defined in Equation 6.3, is correlated with the presence of an object as estimated by the image stream. If the positive density is below a threshold $\rho_t$, the proposal is dropped. We empirically found that $\rho_t = 0.4$ for cars and $\rho_t = 0.3$ for pedestrians produce the best results. The full procedure is summarized in Algorithm 2.

$$\rho(\tilde{x}) = \frac{1}{\Delta} \cdot \sum_{i=-\Delta/2}^{\Delta/2} V(\tilde{x}+i) \tag{6.3}$$

where $V$ is the segmentation vector from the image stream.

**Algorithm 2:** Vector Fusion

**Inputs :** Segmentation Vector $V \in \mathbb{R}^{W' \times K}$
Bird's Eye View Classification Map $C \in \mathbb{R}^{H \times W}$
Classification threshold $\beta$
Positive density threshold $\rho_t$
**Output:** Fused Bird's Eye View Classification Map $\bar{C} \in \mathbb{R}^{H \times W}$

$$\bar{C} \leftarrow \underbrace{\left.\begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}\right\}|H|}_{|W|}$$

**for** $(z, x)$ *in* $C$ **do**
  **if** $C[z, x] \geqslant \beta$ **then**
    $\tilde{x} \leftarrow \frac{f}{f+z} \cdot x + \frac{W'}{2}$
    $\Delta \leftarrow \frac{f}{f+z} \cdot l_a$
    $\rho \leftarrow \frac{1}{\Delta} \cdot \sum_{i=-\Delta/2}^{\Delta/2} V(\tilde{x} + i)$
    **if** $\rho \geqslant \rho_t$ **then**
      $\bar{C}[z, x] \leftarrow C[z, x]$
    **end**
  **end**
**end**

## 6.3 Results

### 6.3.1 Performance on KITTI Benchmark

In this section, we compare the performance of our proposed fusion framework, denoted in the tables as *RGB + LiDAR*, with the deep 3D object detector developed in Chapter 5. The latter serves as a baseline and is hereafter referred to as *LiDAR Only* model. In Table 6.1, we report the average precision of the two models on BEV and 3D detection. The fused model outperforms the baseline with a large margin for pedestrians. These findings are strengthened by the precision-recall curves in Figure 6.5. For the same recall rate, VectorFusion yields a higher precision. However, the benefit of fusion on cars is negligible and in some cases counterproductive. Later experiments reveal that this is a limitation of the segmentation network and not of the fusion framework.

Returning to our main motive of reducing the number of FPs, we compare the classification statistics of our proposed fusion model with the baseline. According to the specifications set in the official KITTI evaluation toolkit, the IoU threshold for a correct detection is 0.7 for cars and 0.5 for pedestrians. As shown in Table 6.2, our fusion method reduces the number of FPs by 32% and 58% for cars and pedestrians respectively. This was coupled with a 15% and 12% increase in FNs for the same classes. All in all, there was a significant increase of 16.48% in the pedestrian $F_1$ score and a modest 2.61% for cars at the common confidence threshold of 0.5.

Table 6.1: Average precision on the KITTI validation set

| | **Modality** | **BEV** | | | **3D** | | |
|---|---|---|---|---|---|---|---|
| | | **E** | **M** | **H** | **E** | **M** | **H** |
| **Car** | **LiDAR Only** | 94.63 | 88.10 | 85.49 | 87.35 | 75.47 | 71.97 |
| | **RGB + LiDAR** | 92.69 | 88.14 | 85.73 | 87.41 | 75.50 | 70.91 |
| | **Delta** | -1.94 | +0.05 | +0.24 | +0.06 | +0.03 | -1.06 |
| **Pedestrian** | **LiDAR Only** | 72.89 | 65.06 | 59.52 | 63.97 | 56.50 | 49.86 |
| | **RGB + LiDAR** | 76.74 | 68.23 | 61.17 | 67.06 | 59.02 | 52.08 |
| | **Delta** | +3.85 | +3.17 | +1.65 | +3.09 | +2.52 | +2.22 |

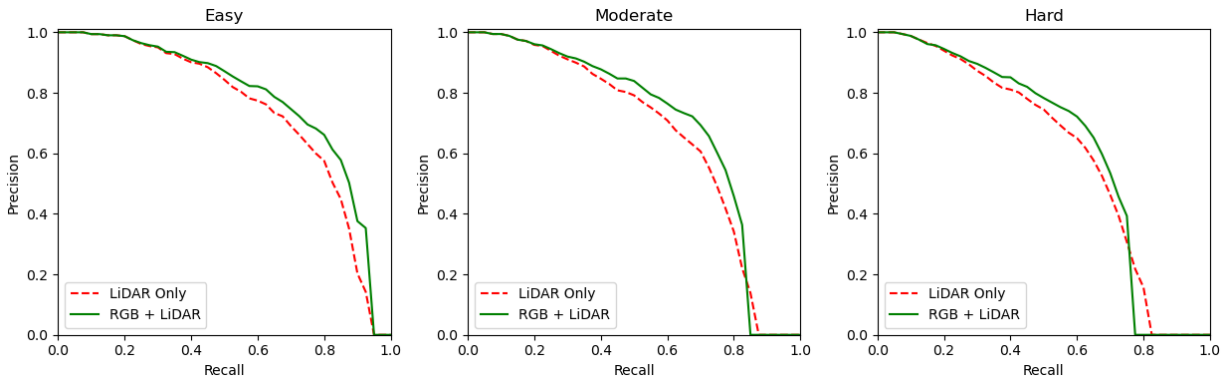



Figure 6.5: Precision-Recall curve for pedestrians in BEV

Table 6.2: The effect of fusion on recall, precision and $F_1$ score

| | Modality | TP | FP | FN | Rec. | Prec. | F1 |
|---|---|---|---|---|---|---|---|
| **Car** | **LiDAR Only** | 8141 | 2799 | 1156 | 87.57 | 74.41 | 80.46 |
| | **RGB + LiDAR** | 7963 | 1912 | 1334 | 85.65 | 80.64 | 83.07 |
| | **Delta** | -178 | -887 | +178 | -1.91 | +6.22 | +2.61 |
| **Pedestrian** | **LiDAR Only** | 1460 | 2625 | 302 | 82.86 | 35.74 | 49.94 |
| | **RGB + LiDAR** | 1421 | 1096 | 341 | 80.65 | 56.46 | 66.42 |
| | **Delta** | -39 | -1529 | +39 | -2.21 | +20.72 | +16.48 |

### 6.3.2 Qualitative Analysis

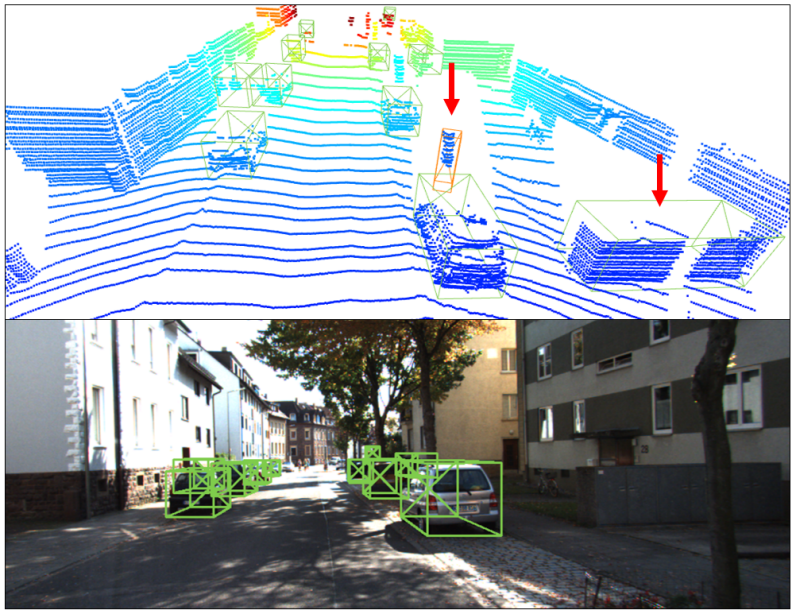
A visual inspection of the detections with and without fusion would give us insight into the model's decision making process. For that reason, we select a few cases and discuss them below.

Out our configuration, the sole purpose of the LiDAR stream is to generate accurate proposals. Ideally, it would have a high recall but low precision. As a result, it is bound to produce a high number of erroneous detections which are later corrected by the fusion module. In Figure 6.6a, the impression of the shipping container in the point cloud resembles that of a car. However, it is easily refuted in the image. Likewise, in both scenes of Figure 6.6, narrow clusters of points are mistaken for pedestrians by the LiDAR stream. All of these false detections were captured and removed by the proposed fusion framework.

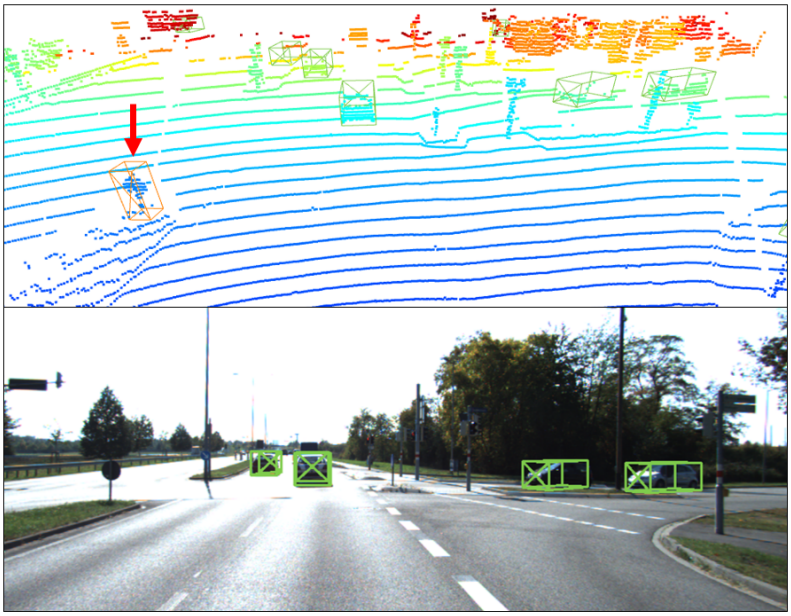
On the contrary, correct detections are prone to removal during fusion if they are not registered by the image stream. We identify two scenarios that degrade the performance of the segmentation network.

1. Obstruction: The pedestrian in Figure 6.7 is barely visible in the image. Yet, it is clearly separated and detectable in the point cloud.

2. Size and Luminance: much like other 2D detection and segmentation networks, the image stream is unable to recognize small and ill-lit objects (See Figure 6.8). In con-



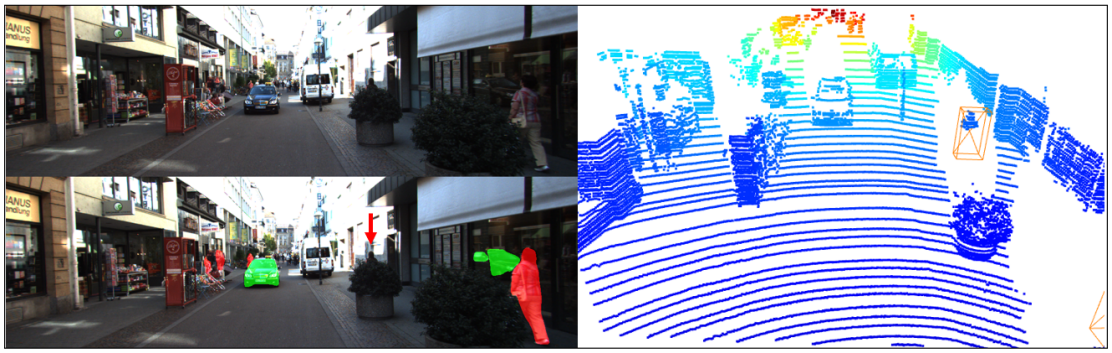
(a)



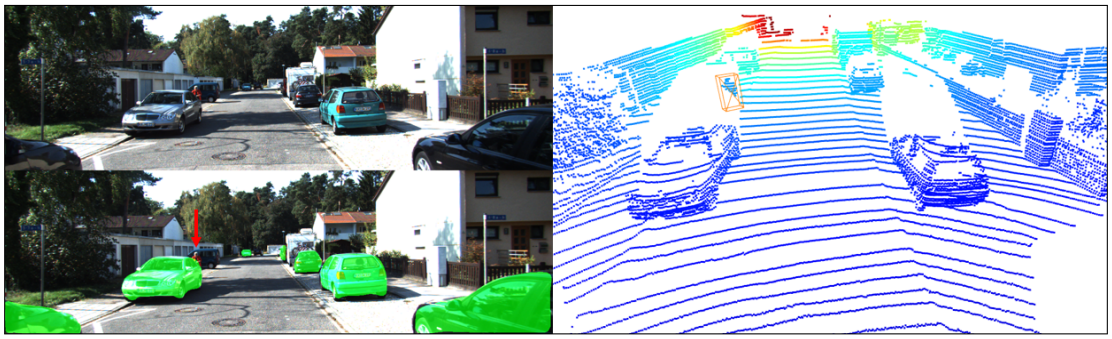
(b)

Figure 6.6: LiDAR FPs rectified through fusion: The objects indicated with a red arrow were incorrectly detected by the LiDAR stream but filtered by the image.

trast, the point cloud isn't affected by variations in size due to the minimal intraclass dimension disparity. Additionally, being an active sensor, the LiDAR is immune to changes in lighting conditions.



(a)

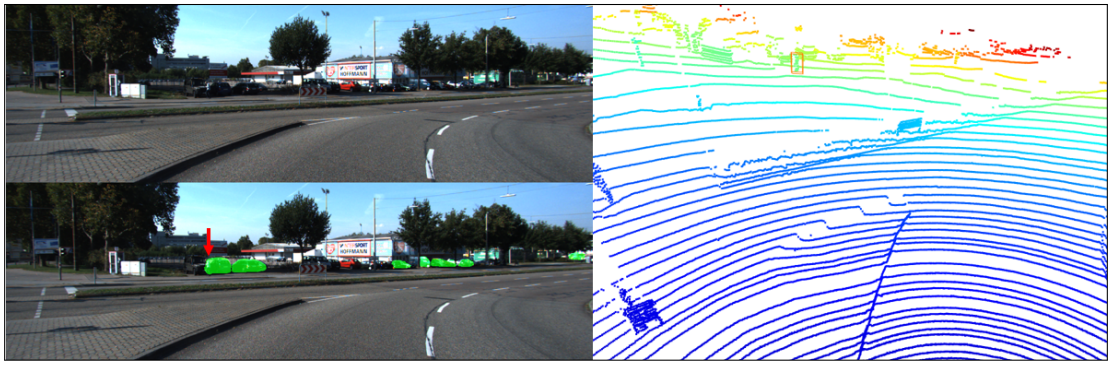


(b)

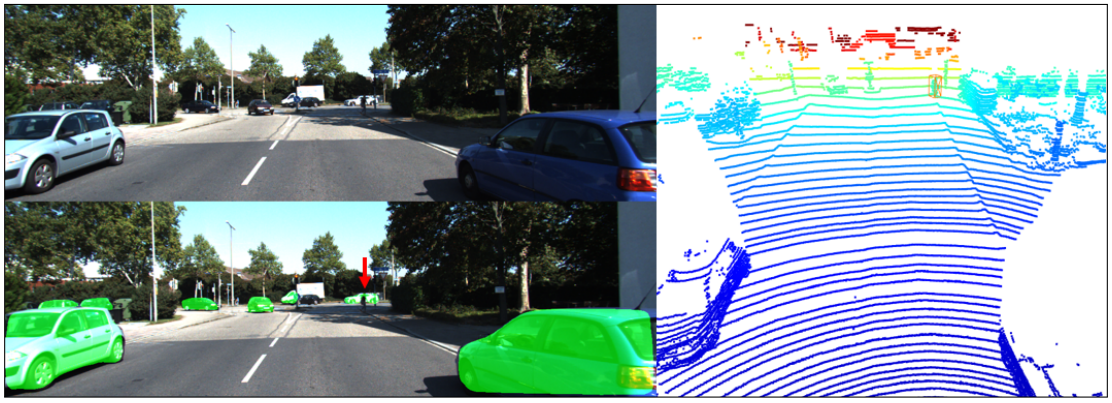
Figure 6.7: Camera failure cases due to obstruction: The pedestrians marked with a red arrow were detected by the point cloud stream but not the image. Top Left - Original Image, Bottom Left - Image with segmentation map overlay, Right - Point cloud detections

### 6.3.3 Effect of Segmentation Stream

The performance of our fusion architecture is dependent on the accuracy of the segmentation network. We therefore quantify its impact on overall performance by replacing it with an immaculate network. As expected, Table 6.3 shows that even more gain can be achieved through a better segmentation network. The biggest increment was seen on pedestrians especially on hard cases. The more compelling find of this experiment, however, is that it proves that the adverse effect of fusion observed previously on cars is caused by the segmentation network.



(a)



(b)

Figure 6.8: Camera failure cases due to size and illumination: The pedestrians marked with a red arrow were detected by the point cloud stream but not the image. Top Left - Original Image, Bottom Left - Image with segmentation map overlay, Right - Point cloud detections

Table 6.3: Impact of segmentation network on BEV and 3D detection performance

| | Segmentation Net | BEV | | | 3D | | |
|---|---|---|---|---|---|---|---|
| | | **E** | **M** | **H** | **E** | **M** | **H** |
| **Car** | **Mask R-CNN [36]** | 92.69 | 88.14 | 85.73 | 87.41 | 75.50 | 70.91 |
| | **Immaculate Seg. Net** | 95.25 | 88.86 | 86.31 | 87.74 | 76.00 | 72.78 |
| **Pedestrian** | **Mask R-CNN [36]** | 76.74 | 68.23 | 61.17 | 67.06 | 59.02 | 52.08 |
| | **Immaculate Seg. Net** | 79.83 | 73.14 | 68.36 | 69.16 | 62.80 | 56.36 |

## 6.4 Vector Fusion, A Generic Framework

Finally, we verify the efficacy of our fusion framework by fusing predictions from other state-of-the art 3D object detectors. Figure 6.9 shows a high level block diagram. The image stream is unaltered but the point cloud stream is replaced with PointPillars [51] and SECOND [102]. Additionally, we further experiment with three point-based methods (PointRCNN [82], Part-$A^2$ [83] and PV-RCNN [81]). The performance on the KITTI validation set for cars is reported in Table 6.4. The results are consistent with previous findings. A marginal change in AP for cars was recorded. In contrast, the performance of all models on the pedestrian class improved substantially as shown in Table 6.5.

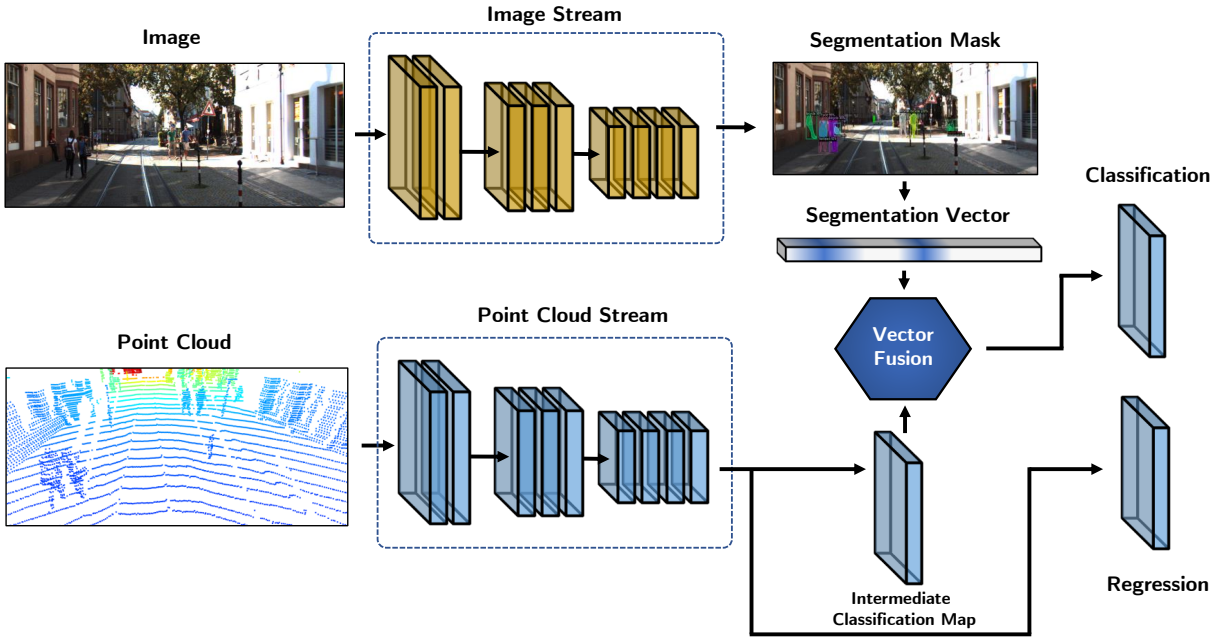



Figure 6.9: High Level Diagram of VectorFusion

Table 6.4: Result of fusing LiDAR based SOTA models on the KITTI Validation Set (Cars)

| | Modality | BEV | | | 3D | | |
|---|---|---|---|---|---|---|---|
| | | **E** | **M** | **H** | **E** | **M** | **H** |
| **PointPillars** [51] | **LiDAR Only** | 91.49 | 87.20 | 85.81 | 86.00 | 75.60 | 72.56 |
| | **RGB + LiDAR** | 90.81 | 86.83 | 84.76 | 85.30 | 75.67 | 72.61 |
| | **Delta** | -0.68 | -0.37 | -1.05 | -0.70 | +0.07 | +0.05 |
| **SECOND** [102] | **LiDAR Only** | 92.04 | 88.04 | 86.65 | 87.75 | 78.40 | 75.19 |
| | **RGB + LiDAR** | 92.44 | 88.66 | 87.65 | 89.14 | 80.07 | 78.47 |
| | **Delta** | +0.40 | +0.62 | +1.00 | +1.39 | +1.67 | +3.28 |
| **PointRCNN** [82] | **LiDAR Only** | 91.39 | 88.28 | 86.31 | 86.65 | 79.30 | 77.26 |
| | **RGB + LiDAR** | 91.49 | 86.47 | 86.22 | 86.71 | 79.28 | 77.23 |
| | **Delta** | +0.10 | -1.81 | -0.09 | +0.06 | -0.02 | -0.03 |
| **Part-A$^2$** [83] | **LiDAR Only** | 92.87 | 89.98 | 88.34 | 92.07 | 82.86 | 81.93 |
| | **RGB + LiDAR** | 93.07 | 88.64 | 88.41 | 90.09 | 82.93 | 80.63 |
| | **Delta** | +0.20 | -1.34 | +0.07 | -1.98 | +0.07 | -1.30 |
| **PV-RCNN** [81] | **LiDAR Only** | 92.95 | 90.28 | 88.50 | 91.94 | 84.25 | 82.41 |
| | **RGB + LiDAR** | 93.01 | 88.76 | 88.54 | 89.88 | 82.96 | 82.25 |
| | **Delta** | +0.06 | -1.52 | +0.04 | -2.06 | -1.29 | -0.16 |

Table 6.5: Result of fusing LiDAR based SOTA models on the KITTI Validation Set (Pedestrians)

| | Modality | BEV | | | 3D | | |
|---|---|---|---|---|---|---|---|
| | | **E** | **M** | **H** | **E** | **M** | **H** |
| **PointPillars** [51] | **LiDAR Only** | 71.21 | 65.55 | 60.22 | 63.36 | 57.78 | 52.10 |
| | **RGB + LiDAR** | 73.58 | 67.28 | 60.68 | 64.97 | 58.94 | 53.35 |
| | **Delta** | +2.37 | +1.73 | +0.46 | +1.61 | +1.16 | +1.25 |
| **SECOND** [102] | **LiDAR Only** | 60.73 | 56.56 | 52.13 | 55.94 | 51.15 | 46.17 |
| | **RGB + LiDAR** | 65.33 | 61.04 | 55.31 | 60.11 | 53.96 | 48.06 |
| | **Delta** | +4.60 | +4.48 | +3.18 | +4.17 | +2.81 | +1.89 |
| **PointRCNN** [82] | **LiDAR Only** | 65.84 | 58.00 | 51.33 | 62.51 | 54.70 | 47.77 |
| | **RGB + LiDAR** | 67.24 | 58.95 | 52.01 | 63.83 | 55.69 | 47.15 |
| | **Delta** | +1.40 | +0.95 | +0.68 | +1.32 | +0.99 | -0.62 |
| **Part-A$^2$** [83] | **LiDAR Only** | 70.53 | 64.20 | 59.24 | 66.88 | 59.68 | 54.61 |
| | **RGB + LiDAR** | 74.39 | 67.64 | 61.06 | 69.52 | 62.74 | 56.10 |
| | **Delta** | +3.86 | +3.44 | +1.82 | +2.64 | +3.06 | +1.49 |
| **PV-RCNN** [81] | **LiDAR Only** | 65.93 | 58.48 | 54.13 | 62.71 | 54.46 | 49.86 |
| | **RGB + LiDAR** | 71.54 | 63.81 | 57.56 | 68.20 | 59.31 | 54.24 |
| | **Delta** | +5.61 | +5.33 | +3.43 | +5.49 | +4.85 | +4.38 |

## 6.5 Conclusion

In this chapter, we presented a late fusion scheme that uses segmentation maps generated from an RGB image to complement the classification head of FA3D and other recent 3D object detectors to reduce the number of false detections. Our experiments show that the average precision increases by up to 3.85 AP points when using our method on FA3D.

# Chapter 7

# Conclusion

## 7.1 Conclusion

Advanced driver assistance systems, and subsequently autonomous vehicles, are exciting technological breakthroughs that have the potential to improve the safety and quality of driving. Robust and timely perception is perhaps the most important component of these intelligent systems. In Chapter 5 of this thesis, we introduced a fast and accurate deep learning model that detects 3D objects from point clouds. Our proposed network utilizes a compute and bandwidth efficient, yet expressive input representation that considerably decreases the time elapsed for encoding and transferring the point cloud. One of the main advantages of our solution is that we do not gain time improvement by designing a smaller or more biased network. Our model, as it happens, is complex and highly flexible. As a result, the presented detector improved the frame rate of the state-of-the-art model by 18% and 43% on a Jetson Xavier and a high-end workstation respectively. We were also able to demonstrate the practicality of our model by deploying it on a real-world embedded vision processor from Synopsys. In addition to the detailed accuracy and runtime experiments, we methodically analyzed the predictions and quantified the impact of each regression target on average precision.

Nowadays, autonomous vehicles are equipped with a suite of exteroceptive sensors to ensure robust perception of the physical world. Particularly, camera-LiDAR fusion

has gained traction in recent years due to the sensors' complementary nature. LiDARs generate sparse information that lacks color and texture information while images deliver ambiguous 3D location and dimension. In Chapter 6, standing on finding from the error analysis section of FA3D, we proposed a late fusion scheme that uses segmentation maps generated from RGB images to complement the classification head. We evaluated the model on the KITTI benchmark and reported improvements in pedestrian detection. On the other hand, there was minimal gain in the accuracy of vehicle detection due to the segmentation network. Additionally, we interpreted the model's predictions by visually inspecting different cases and identified conditions were true positives were incorrectly removed. At last, we tested the proposed strategy on other state-of-the-art detectors and confirmed similar gains.

## 7.2 Future Directions

**Evaluation Metrics** Most of the efforts in 3D object detection have been directed towards either increasing average precision or reducing latency. Even though this motive has pushed the research this far, its impact on overall safety of the vehicle is not clear due to two reasons. Firstly, current algorithms are assessed against ground truth boxes collected when the frame was captured. By the time the algorithm processes the incoming point cloud and renders a list of detections, additional ground truth boxes might enter the field of view and existing boxes might move to another location or exit the scene. This raises questions about the validity of evaluations done on stale ground truth boxes. Secondly, average precision for 3D object detection was adapted from the 2D literature where equal weight is allocated to all objects irrespective of location or size. Thus, we argue that average precision, in its current form, is a suboptimal metrics for autonomous vehicles because some objects are more important than others. For example, a human driver focuses more on closer objects and objects along the ego-vehicle's trajectory. For these reasons, further research should be conducted to find metrics that are better correlated to the challenge at hand.

**End-to-End Point Cloud Perception** In essence, the task of detecting 3D bounding boxes from point clouds can be reduced to a set-to-set transformation. The input is a set of points while the output is a set of bounding boxes. Therefore, a truly end-to-end pipeline will consume the point cloud in its raw form without voxelizing or grouping and yield detections without postprocessing. Permutation invariant deep learning architectures such as [106] are potential starting points. In addition, transformers [94], without positional encoding, can also be used for predicting one set from another. Recently, DETR [12] demonstrated a similar idea of using transformers to detect objects from images. They were able to streamline the problem by parting with engineered aspects of object detection such as anchors and non-maximum suppression. One apparent challenge in using transformers for point clouds is the quadratic growth in both memory and computation time of the attention module with respect to the length of the input (number of points).

**Joint Augmentation** Monomodal detectors, including FA3D, have recorded substantial gains in accuracy through the use of augmentation. The same cannot be said for multimodal detectors as it has been very difficult to coherently augment images and point clouds together. For instance, rotating a vehicle in the point cloud will reveal occluded parts of the car which are very hard to reconstruct in the image. This is an exciting research problem and perhaps an ideal candidate for generative adversarial networks (GANs) [32].

**Diverse Datasets and Challenges** Existing works are trained and evaluated on narrow driving conditions that might not accurately reflect variations encountered in the real world. In this regard, training on diverse datasets and investigating the effects of lower ambient lighting and anomalous weather conditions such as fog, rain and snow is an open research opportunity. In addition, much like other deep learning models, 3D perception pipelines are susceptible to attacks. Thus, identifying and countering natural and intentional adversarial attacks from the surrounding is also an interesting area to explore.

# References


[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN, 2017.

[3] Edward H. Adelson, Peter J. Burt, Charles H. Anderson, Joan M. Ogden, and James R. Bergen. Pyramid Methods in Image Processing. *RCA engineer*, 29(6):33–41, 1984.

[4] National Highway Traffic Safety Administration. Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812506, 2018.

[5] Hamed H Aghdam, Elnaz J Heravi, Selameab S Demilew, and Robert Laganière. RAD: Realtime and Accurate 3D Object Detection on Embedded Systems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2875–2883, 2021.

[6] Waleed Ali, Sherif Abdelkarim, Mahmoud Zidan, Mohamed Zahran, and Ahmad El Sallab. YOLO3D: End-to-end real-time 3D oriented object bounding box detection from LiDAR point cloud. *Lecture Notes in Computer Science (including subseries*

*Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11131 LNCS:716–728, 2019.

[7] Alejandro Barrera, Carlos Guindel, Jorge Beltrán, and Fernando García. BirdNet+: End-to-End 3D Object Detection in LiDAR Bird's Eye View. 2020.

[8] Jorge Beltrán, Carlos Guindel, Francisco Miguel Moreno, Daniel Cruzado, Fernando García, and Arturo De La Escalera. BirdNet: A 3D Object Detection Framework from LiDAR Information. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018-Novem:3517–3523, 2018.

[9] Robert C Bolles and Martin A Fischler. A ransac-based approach to model fitting and its application to finding cylinders in range data. In *IJCAI*, volume 1981, pages 637–643, 1981.

[10] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, and Qiang Xu. nuScenes : A multimodal dataset for autonomous driving. (March), 2019.

[11] Statistics Canada. Vehicle registrations, by type of vehicle. https://www150.statcan.gc.ca/t1/tbl1/en/tv.action?pid=2310006701, 2020.

[12] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-End Object Detection with Transformers.

[13] Jia Ren Chang and Yong Sheng Chen. Pyramid Stereo Matching Network. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018.

[14] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D object detection network for autonomous driving. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 6526–6534, 2017.

[15] Yilun Chen, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Fast point R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob:9774–9783, 2019.

[16] Corinna Cortes and Vladimir Vapnik. Support vector machine. *Machine learning*, 20(3):273–297, 1995.

[17] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. *Advances in Neural Information Processing Systems*, pages 379–387, 2016.

[18] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[19] Selameab S Demilew, Hamed H Aghdam, Robert Laganière, and Emil M Petriu. FA3D: Fast and Accurate 3D Object Detection. In *International Symposium on Visual Computing*, pages 397–409. Springer, 2020.

[20] Selameab S Demilew, Hamed H Aghdam, and Robert Laganière. Refine 3D Bounding Boxes through Late Vector Fusion for ADAS Applications. In *4th BAI Workshop, NeurIPS, 2020*.

[21] Martin Dillon. Introduction to modern information retrieval: G. salton and m. mcgill. mcgraw-hill, new york (1983). xv+ 448 pp., 32.95 isbn 0-07-054484-0, 1983.

[22] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[23] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge 2007 (voc2007) results. 2007.

[24] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008.

[25] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep Ordinal Regression Network for Monocular Depth Estimation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018.

[26] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research. *The International Journal of Robotics Research*, (October):1–6, 2013.

[27] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[28] Golnaz V Ghiasi Google Brain Tsung-Yi Lin Google Brain Quoc Le Google Brain. DropBlock. (NeurIPS):1–11, 2018.

[29] Ross Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1440–1448, 2015.

[30] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[31] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[32] Ian J Goodfellow, Jean Pouget-abadie, Mehdi Mirza, Bing Xu, and David Warde-farley. Generative Adversarial Nets. pages 1–9.

[33] K. Habib. Technical report, Tesla crash. https://static.nhtsa.gov/odi/inv/2016/INCLA-PE16007-7876.PDF, 2017.

[34] Kazuyuki Hara, Daisuke Saitoh, and Hayaru Shouno. Analysis of dropout learning regarded as ensemble learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9887 LNCS:72–79, 2016.

[35] Chenhang He, Hui Zeng, Jianqiang Huang, Xian-sheng Hua, and Lei Zhang. Structure Aware Single-stage 3D Object Detection from Point Cloud. *IEEE Conference on Computer Vision and Pattern Recognition*, 1, 2020.

[36] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):386–397, 2020.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:770–778, 2016.

[38] Yuanduo Hong, Huihui Pan, Weichao Sun, Senior Member, IEEE, and Yisong Jia. Deep Dual-resolution Networks for Real-time and Accurate Semantic Segmentation of Road Scenes. 14(8):1–12, 2021.

[39] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:3296–3305, 2017.

[40] Apple Inc. Apple unveils new iPad Pro with breakthrough LiDAR Scanner and brings trackpad support to iPadOS. https://www.apple.com/ca/newsroom/2020/03/apple-unveils-new-ipad-pro-with-lidar-scanner-and-trackpad-support-in-ipados, 2020.

[41] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2015.

[42] Michael Jones and Paul Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, 3(14):2, 2003.

[43] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[44] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollar. Panoptic feature pyramid networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:6392–6401, 2019.

[45] Tao Kong, Anbang Yao, Yurong Chen, and Fuchun Sun. HyperNet: Towards accurate region proposal generation and joint object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:845–853, 2016.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[47] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[48] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L. Waslander. Joint 3D Proposal Generation and Object Detection from View Aggregation. *IEEE International Conference on Intelligent Robots and Systems*, pages 5750–5757, 2018.

[49] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *IJCV*, 2020.

[50] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[51] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:12689–12697, 2019.

[52] Johannes Lehner, Andreas Mitterecker, Thomas Adler, Markus Hofmarcher, Bernhard Nessler, and Sepp Hochreiter. Patch Refinement – Localized 3D Object Detection. (NeurIPS 2019), 2019.

[53] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3D lidar using fully convolutional network. *Robotics: Science and Systems*, 12, 2016.

[54] Ming Liang, Yang Bin, Yun CHen, Rui Hu, and Raquel Urtaun. Multi-Task Multi-Sensor Fusion for 3D Object Detection. 2019.

[55] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep Continuous Fusion for Multi-sensor 3D Object Detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11220 LNCS:663–678, 2018.

[56] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *Proceedings. international conference on image processing*, volume 1, pages I–I. IEEE, 2002.

[57] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[58] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125. IEEE, 11 2017.

[59] Tsung Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020.

[60] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8693 LNCS(PART 5):740–755, 2014.

[61] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS:21–37, 2016.

[62] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[63] Gregory P. Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K. Wellington. Lasernet: An efficient probabilistic 3D object detector for autonomous driving. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:12669–12678, 2019.

[64] Arsalan Mousavian, Dragomir Anguelov, Jana Košecká, and John Flynn. 3D bounding box estimation using deep learning and geometry. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:5632–5640, 2017.

[65] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[66] Jiquan Ngiam, Benjamin Caine, Wei Han, Brandon Yang, Yuning Chai, Pei Sun, Yin Zhou, Xi Yi, Ouais Alsharif, Patrick Nguyen, Zhifeng Chen, Jonathon Shlens, and Vijay Vasudevan. StarNet: Targeted Computation for Object Detection in Point Clouds. 2019.

[67] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, 2015.

[68] Nutonomy. second.pytorch. https://github.com/nutonomy/second.pytorch, 2019.

[69] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[70] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

[71] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:77–85, 2017.

[72] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016.

[73] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems*, 2017-Decem:5100–5109, 2017.

[74] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:779–788, 2016.

[75] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:6517–6525, 2017.

[76] Joseph Redmon and Ali Farhadi. YOLO v.3. *Tech report*, pages 1–6, 2018.

[77] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[78] Thomas Roddick, Alex Kendall, and Roberto Cipolla. Orthographic Feature Transform for Monocular 3D Object Detection. 2018.

[79] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[80] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.

[81] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection. 2019.

[82] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D object proposal generation and detection from point cloud. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:770–779, 2019.

[83] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From Points to Parts: 3D Object Detection from Point Cloud with Part-aware and Part-aggregation Network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020.

[84] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-YOLO: Real-time 3D Object Detection on Point Clouds. pages 1–14, 2018.

[85] Andrea Simonelli, Samuel Rota Bulo, Lorenzo Porzi, Manuel Lopez-Antequera, and Peter Kontschieder. Disentangling monocular 3D object detection. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob:1991–1999, 2019.

[86] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, 2015.

[87] John Sousanis. World Vehicle Population Tops 1 Billion Units. http://wardsauto.com/ar/world_vehicle_population_110815/, 2011.

[88] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[89] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[90] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan,

Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Sheng Zhao, Shuyang Cheng, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. 2019.

[91] Synopsys. DesignWare EV6x Vision Processors. https://www.synopsys.com/dw/ipdir.php?ds=ev6x-vision-processors, 2020.

[92] Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. 2019.

[93] Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. pages 10778–10787, 2020.

[94] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):5999–6009, 2017.

[95] Sourabh Vora, Alex H. Lang, Bassam Helou, and Oscar Beijbom. PointPainting: Sequential Fusion for 3D Object Detection. 2019.

[96] Yan Wang, Wei-lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-LiDAR from Visual Depth Estimation : Bridging the Gap in 3D Object Detection for Autonomous Driving. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8445–8453, 2019.

[97] Zhixin Wang and Kui Jia. Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal. *IEEE International Conference on Intelligent Robots and Systems*, pages 1742–1749, 2019.

[98] Zining Wang, Wei Zhan, and Masayoshi Tomizuka. Fusing Bird's Eye View LIDAR Point Cloud and Front View Camera Image for 3D Object Detection. *IEEE Intelligent Vehicles Symposium, Proceedings*, 2018-June:834–839, 2018.

[99] Steven Waslander and Jonathan Kelly. Self-Driving Cars Specialization. https://www.coursera.org/specializations/self-driving-cars, 2020.

[100] World Health Organization (WHO). Global Status Report on Road Safety. https://www.who.int/violence_injury_prevention/road_safety_status/2018/en/, 2018.

[101] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:1912–1920, 2015.

[102] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors (Switzerland)*, 18(10):1–17, 2018.

[103] Bin Yang, Wenjie Luo, and Raquel Urtasun. PIXOR: Real-time 3D Object Detection from Point Clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[104] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. IPOD: Intensive Point-based Object Detector for Point Cloud. 2018.

[105] Yurong You, Yan Wang, Wei-Lun Chao, Divyansh Garg, Geoff Pleiss, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving. *arXiv preprint arXiv:1906.06310*, 2019.

[106] Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. *Advances in Neural Information Processing Systems*, 2017-Decem(ii):3392–3402, 2017.

[107] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as Points. 2019.

[108] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:5738–5746, 2019.

[109] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.