Real-time 3D Object Detection for Autonomous Driving

by

Melissa Mozifian

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2018

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis focuses on advancing the state-of-the-art 3D object detection and localization in autonomous driving. An autonomous vehicle requires operating within a very unpredictable and dynamic environment. Hence a robust perception system is essential. This work proposes a novel architecture, AVOD, an **A**ggregate **V**iew **O**bject **D**etection architecture for autonomous driving capable of generating accurate 3D bounding boxes on road scenes. AVOD uses LIDAR point clouds and RGB images to generate features that are shared by two subnetworks: a region proposal network (RPN) and a second stage detector network.

The proposed RPN uses a novel architecture capable of performing multimodal feature fusion on high resolution feature maps to generate reliable 3D object proposals for multiple object classes in road scenes. Using these proposals, the second stage detection network performs accurate oriented 3D bounding box regression and category classification to predict the extents, orientation, and classification of objects in 3D space.

AVOD is differentiated from the state-of-the-art by using a high resolution feature extractor coupled with a multimodal fusion RPN architecture, and is therefore able to produce accurate region proposals for small classes in road scenes. AVOD also employs explicit orientation vector regression to resolve the ambiguous orientation estimate inferred from a bounding box.

Experiments on the challenging KITTI dataset show the superiority of AVOD over the state-of-the-art detectors on the 3D localization, orientation estimation, and category classification tasks. Finally, AVOD is shown to run in real time and with a low memory overhead. The robustness of AVOD is also visually demonstrated when deployed on our autonomous vehicle operating under low lighting conditions such as night time as well as in snowy scenes.

Furthermore, AVOD-SSD is proposed as a 3D Single Stage Detector. This work demonstrates how a single stage detector can achieve similar accuracy as that of a two-stage detector. An analysis of speed and accuracy trade-offs between AVOD and AVOD-SSD are presented.

Acknowledgements

I would like to thank my supervisor, Professor Steven Waslander, for providing me with the opportunity to explore many great projects and ideas. His guidance, encouragement, and support have been pivotal in the success of my work. I would like to also thank all the people who made this thesis possible including my wonderful colleagues Jason Ku and Ali Harakeh, and other members of the Waterloo Autonomous Vehicle Laboratory including Nima Mohajerin, Christopher Choi, Stanley Brown and many more. They have been an absolute pleasure to work with and have helped me greatly throughout my Masters journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The detection and localization of objects within an environment is a key problem in creating truly autonomous cars. The knowledge of the positions of other cars, pedestrians and cyclists is required to safely maneuver an autonomous vehicle on the road.

Object detection is a challenging problem, in particular for autonomous vehicles since a system must be able to detect objects reliably within the environment in various conditions such as low lighting conditions and snow. In addition to high accuracy detection, it also needs to be robust and have minimal false positive detections. These properties are crucial as other modules on the car such as the behavioral planner, make decisions based on what is detected within the environment.

Deep convolutional neural networks (CNNs) have revolutionized object detection. They have been widely used in visual recognition since 2012 [40], when they outperformed existing methods on the task of image classification on the ImageNet dataset [62] by a large margin. Since then CNNs have been widely used for tackling visual recognition problems.

Most of the research focus in recent years has been on the task of improving the accuracy and speed of 2D object detectors. Modern successful object detectors such as Faster R-CNN [60], R-FCN [7], Multibox [69] and YOLO [57] classify objects with a 2D bounding box in image space. These networks are performing well enough to be deployed in consumer products. However in the context of the 3D world and the task of autonomous driving, 2D detection does not capture all the required information such as depth and orientation from the surrounding environment. Also the performance of 2D detectors has not transferred well to the detection of objects in 3D. The gap between the two remains large on standard benchmarks such as the KITTI Object Detection Benchmark [23], where 2D *car* detectors such as RRC [59] have achieved over 90% Average Precision (AP). In contrast to 2D, the

top scoring 3D *car* detector on the same data only achieves around 70% 3D AP and 84% Bird's Eye View (BEV) AP. The reason for such a gap stems from the difficulty induced by adding a third dimension to the estimation problem, the low resolution of 3D input data, and the deterioration of its quality as a function of distance. Furthermore, unlike 2D object detection, the 3D object detection task requires estimating *oriented* bounding boxes.

Region proposal networks (RPNs) were proposed in Faster-RCNN [60], and have become the prevailing technique towards improving the accuracy of 2D detectors. RPNs can be considered a weak amodal detector, providing proposals with high recall and low precision. An RPN is essentially a network hypothesizing the location of potential objects, by proposing a set of regions that are likely to contain objects.

Single stage detectors such as YOLO [57] on the other hand, propose an RPN-free architecture to perform detection at a higher speed but with lower accuracy. Extending both these techniques to 3D to achieve the same accuracy and speed is a non-trivial task. 2D detectors are typically tailored for dense, high resolution image input, where objects usually occupy a relatively large portion of pixels in the feature map. When considering sparse and low resolution input such as the Front View [44] or BEV [6] point cloud projections, these methods are not guaranteed to contain sufficient information to perform the detection with the same accuracy.

Similar to 2D object detectors, most state-of-the-art models for 3D object detection rely on a *3D region proposal generation* step for 3D search space reduction. Using region proposals allows the generation of high quality detections via more complex and computationally expensive processing at later detection stages. However, any missed instances at the proposal generation stage cannot be recovered during the following stages. Therefore, achieving a high recall during the the region proposal generation stage is crucial to achieve good performance. Most current 3D object detection methods leverage multimodal input from cameras and 3D range sensors. Methods using LIDAR point clouds [6, 44, 43, 15], stereo depth maps [5], or RGBD sensor depth maps [67], have been shown to outperform image only methods [3, 50, 2] on the 3D object detection task. Multimodal object detectors have been previously discussed in the literature [28, 24, 53, 16, 6, 5]. MV3D [3] proposed an RPN network that generates 3D candidate boxes from the BEV representation of 3D point cloud. No previous work however, has exploited fusing features from both point cloud and RGB data to generate region proposals. Both sources of information are crucial for generating better region proposals, and in particular, lead to higher performance when targeting small object classes such as pedestrians and cyclists.

This thesis proposes AVOD, an **A**ggregate **V**iew **O**bject **D**etection architecture for

autonomous driving. AVOD uses LIDAR point clouds and RGB images to generate features that are shared by two subnetworks: a region proposal network (RPN) and a second stage detector network. The RPN is capable of using features from multiple input modes for the proposal generation task, allowing for higher recall proposal generation on smaller objects in road scenes. Using these proposals, the second stage detection network performs accurate *oriented* 3D bounding box regression predicting the extents, orientation, and category classification of objects in 3D. AVOD aims to solve the following limitations in current state-of-the-art 3D object detection methods:

1. Improvements over MV3D [6] in terms of detecting smaller objects such as pedestrian and cyclists using the fusion mechanism. Furthermore, fusion of image and BEV data, avoids relying on one source of information.

2. More robust heading estimation using the proposed box representation as opposed to 8-corner representation used in MV3D.

3. Speed improvement over the state-of-the-art 3D detectors.

Item 1 is concerned with the detection of smaller sized objects. The Faster R-CNN RPN architecture is tailored for dense, high resolution image input, where objects usually occupy more than one pixel in the feature map. When considering sparse and low resolution input such as the BEV point cloud projection generated from LIDAR, this method is not guaranteed to have enough information from a single pixel to generate region proposals. As an example, an average pedestrian in the KITTI dataset occupies 0.8×0.6 meters in the BEV. This translates to an 8×6 pixel area in a BEV map with 0.1 meter resolution. When downsampled by convolutional feature extractors, instances from this class will occupy a fraction of a pixel in the resultant feature map. Even with higher resolution BEV maps, the sparsity of LIDAR data is not sufficient to extract informative features for these small sized classes. This is one short-coming in MV3D [6], where only BEV map is used to generate proposals. The authors reported detecting small objects as a difficult problem with their architecture and only reported results on the task of car detection.

AVOD aims to fuse features from the image and the BEV feature maps as inputs to the RPN, allowing the generation of high recall proposals for smaller classes. Item 1 is also concerned with relying on 2D detectors for initial hypothesis as potential objects in the environment as proposed in architectures such as F-PointNet [54]. Section 3.3 demonstrates that AVOD is robust to noisy LIDAR data and lighting changes affecting the image data, as it was tested in snowy scenes and in extreme low light conditions. This is due to the fusion

mechanism of both sources of information at the RPN stage, allowing reliable proposal generation even when LIDAR or image quality are degraded.

Item 2 addresses another shortcoming of MV3D in terms of its heading estimation mechanism that relies on extracting heading information from the regressed 8 corners of a 3D bounding box. Section 3.3 demonstrates that this method cannot be used to extract unique orientation information from only the corners of the bounding box. AVOD explicitly regresses an orientation vector that is used to correct the orientation estimate extracted from such box representations. This is shown to provide a 29.46% increase in AHS (explained in Section 3.2.1) over MV3D on the Car class of the KITTI dataset.

Item 3 is concerned with the speed of state-of-the-art networks. AVOD has been optimized for speed and is currently the fastest two-staged 3D detector compared to other state-of-the-art 3D detectors, as shown in Table 3.6.

AVOD has been evaluated on the tasks of proposal generation, 3D detection, and BEV detection on the challenging KITTI 3D object detection benchmark. AVOD has also been demonstrated to work in snow and at night due to the proposed feature fusion from both image and LIDAR sources, making it a suitable candidate for deployment on autonomous vehicles.

This work further explores the impacts of modifying AVOD to operate as a Single Stage Detector, using existing techniques to improve the performance of the network as an object classifier and bounding box regressor. The advantage of a single stage detector is the simplicity in the architecture as well as reduction in the number of network parameters. Although existing 2D single stage detectors operate at relatively high speed compared to two-stage 2D detectors, there are currently no existing fast single stage 3D detectors.

AVOD is a joint effort with my colleagues Jason Ku and Ali Harakeh. The original AVOD architecture inspired by MV3D [6] was proposed by Ali Harakeh. Customizing the feature extractor to improve detection on pedestrian and cyclist was proposed by myself, which led to using Feature Pyramid on AVOD to boost performance. The core implementation of AVOD as well as deployment of AVOD on our autonomous vehicle was done by myself and Jason Ku. The work on AVOD-SSD was done by myself with the aim to understand if single stage detectors can match or surpass the accuracy of AVOD while running at similar or faster speed. All code is implemented using Google's Tensorflow [17] and Robot Operating System (ROS) for deployment on the car [56].

The remainder of this thesis is organized as follows: Chapter 2 provides some background on deep learning and in particular object detection methods. Chapter 3 presents the article containing new methods. Chapter 4 presents a single stage 3D object detector

and the speed and accuracy trade-off of two-stage vs. single stage are studied. Finally chapter 5 concludes this thesis [1].

Resources

- AVOD arxiv paper can be found at https://arxiv.org/abs/1712.02294
- The AVOD code can be found at https://github.com/kujason/avod
- The AVOD-SSD code can be found at https://github.com/melfm/avod-ssd

More qualitative results including those of AVOD running in **snow** and **night** scenes are provided in video format here.

[1] Throughout this thesis, the narrator will be referred to as "we", rather than "I". This is because the AVOD research presented in this thesis was developed in a collaborative setting. My specific contributions are outlined in the introduction, and all writing outside of Chapter 3 on AVOD is entirely my own.

Chapter 2

Background

This chapter briefly presents the background information required for the contributions made by this work. This includes a brief overview of Neural Networks and Convolutional Neural Networks in particular, as well as providing some background on object detection methods and current state-of-the-art techniques under both 2D and 3D settings.

2.1 Feedforward Neural Networks

The Feedforward Neural Networks (FNN), are the most basic and common type of artificial neural networks deployed in practical applications. The goal of a feedforward network is to approximate some function f^*. A feedforward network defines a mapping $y = f(x;\theta)$ where x is the input and f is a function mapping x to some output y. A feedforward neural network with l hidden layers is parameterized by $l+1$ weight matrices $(W_0, ..., W_l)$ and $l+1$ vectors of biases $(b_1, ..., b_{l+1})$. The concatenation of the weight matrices and the biases forms the parameter vector θ, which specifies the function computed by the network. The network learns the value of the parameters θ that result in the best function approximation. Feedforward neural networks are called networks because they are typically represented by composing together many different functions. A neural network can be associated with a directed acyclic graph describing how the functions at each layer are composed. For example, we could have three functions $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ connected in a chain resulting to $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ where $f^{(1)}$ is the first layer, $f^{(2)}$ is the second layer and so on. The final layer of a feedforward network is called the output layer [29]. A neural network learns certain features in an input, where features can be variables or attributes in the data. For instance in the case of images, such features are learned in an hierarchical

fashion, where the low level features represent edges, boundaries and lines, and as we go deeper into the hierarchy, the features become much more complex representing shapes and attributes associated with the target object. A neural network essentially learns to look for certain features, and when it is shown an unseen sample, it calculates the probabilities for the features it is trained to look for.

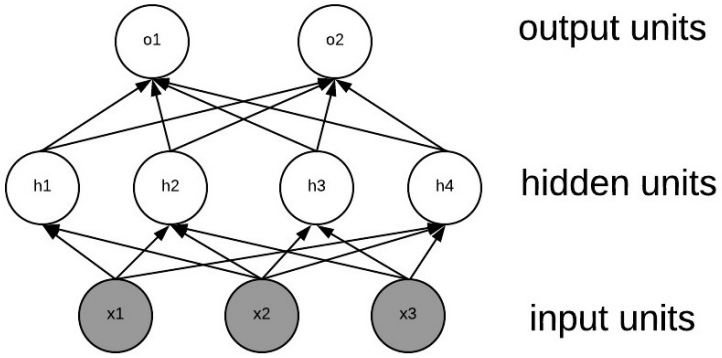


Figure 2.1: Feedforward neural network with inputs x_1, x_2, x_3, four hidden units h_1, h_2, h_3, h_4 and two output units o_1, o_2.

2.2 Convolutional Networks

Convolutional neural networks [42], also known as or CNNs or ConvNets, are a kind of neural network designed to process data that has a known grid-like topology [29]. Convolutional networks have been extremely successful in practical image processing applications. Convolutional networks are simply neural networks that apply the convolution operation in place of general matrix multiplication in at least one of their layers.

2.2.1 The Convolution Operation

Convolution is a mathematical operation on two functions f and g to produce a third function. Convolution is defined as an integral that expresses the amount of overlap of g as it is shifted over f. The convolution operation is typically denoted with an asterisk:

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

The convolution formula can be described as a weighted average of the function $f(\tau)$ at the moment t where the weighting is given by $g(-\tau)$ shifted by the amount of t. As t changes, the weighting function g places more emphasis on different parts of the input function f. Note that in general, convolution is defined for any function for which the above integral is defined.

In convolutional network terminology, function f to the convolution is often referred to as the input, and function g is the kernel. The output is sometimes referred to as the feature map. When working with data on a computer, time and space usually will be discretized. Therefore we can define the discrete convolution, assuming f and g are defined only on integer t :

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau) \tag{2.2}$$

In the context of convolutional neural networks, the input is usually a multidimensional array of data, and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. These multidimensional arrays are referred to as tensors.

We often use convolution over more than one axis at a time. For instance in the case of a 2D image I as input, we would also use a two-dimensional kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(m,n)K(i-m, j-n) \tag{2.3}$$

where S is the output or feature map [29].

Convolution leverages three important ideas that can help improve a machine learning system: **sparse interactions**, **parameter sharing** and **equivariant representations**.

A fully connected neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This way, as demonstrated in Figure 2.1, every output unit interacts with the input unit. However, it is impractical to connect neurons to all neurons when dealing with high-dimensional inputs such as images and hence, fully connected networks don't scale well to full images. Convolutions deploy a kernel of a size smaller than the input, and this enables them to connect each neuron to only a local region of the input volume.

Convolutional networks efficiency in comparison to traditional fully connected networks are quite significant. If there are m inputs and n outputs, then matrix multiplication requires $m \times n$ parameters, and the algorithms used in practice have $O(m \times n)$ runtime per training sample. If we limit the number of connections from each output to k, then the sparsely connected approach requires only $k \times n$ parameters and $O(k \times n)$ runtime.

For example, consider the task of processing an image, where the input might have thousands or millions of pixels. We can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels. With convolutions, we need to store fewer parameters. This in turn reduces the computational demand of the network.

Parameter sharing in convolutional networks occurs since each member of the kernel is used at every position of the input and hence instead of learning a separate set of parameters for every location, the network learns only one set of parameters. This is in contrast to a fully connected network setting where each element of the weight matrix is used exactly once when computing the output of a layer.

Finally in the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation. Specifically, a function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$. In the case of images, convolution creates a $2D$ map of where certain features appear in the input. Therefore if we move the object in the input, its representation will move the same amount in the output [29].

2.2.2 Convolutional Network Structure

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. Typically three types of layers are used to build ConvNet architectures: convolutional layer, pooling layer, and fully-connected (FC) layer, where in a fully connected layer neurons have full connections to all activation from previous neurons, as seen in regular FNNs. These layers are stacked to form a ConvNet architecture.

The convolution layer computes the output of neurons that are connected to local regions in the input, by computing a dot product of weights and a small region they are connected to in the input volume to produce a set of linear activations. Next, each linear activation is run through a nonlinear activation function. In the next stage, the pooling layer performs a downsampling operation along the spatial dimensions (width, height). Hence it is common to periodically insert a pooling layer in-between successive convolution layers. Finally the FC layer computes the class scores.

The spatial arrangement of a Conv layer output volume is controlled by three hyperparameters: the **depth**, **stride** and **zero-padding**. The depth corresponds to the number of filters the network uses to look for different features in the input. The stride specifies how to slide the filter. When the stride is 1, the filter is moved one pixel at a time, and when the stride is 2, the filter jumps two pixels, and so on. This, in turn will produce a smaller output volume spatially. The padding is used to pad the input volume with zeros, which enables the network to control the spatial size of the output volumes.

2.2.3 Receptive Field

ConvNets enable connecting each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the **receptive field** of the neuron, i.e. filter size. For instance, Figure 2.2 shows an input volume of the size $[4 \times 4]$. If the receptive field is 2×2 then each neuron in the convolution layer will have weights to a $[2 \times 2]$ region in the input volume, a total of $2 * 2 = 4 + 1$ weights (where $+1$ is the bias parameter).

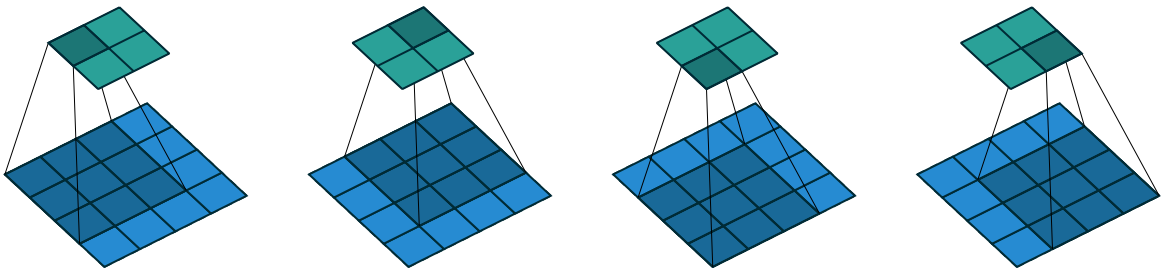


Figure 2.2: Concept of receptive field visualized with an example of convolving a 2×2 kernel over a 4×4 input using unit strides [14].

2.2.4 Pooling

A pooling function replaces the output of the network at a certain location with a summary statistic of the nearby outputs. For instance **max pool** gives the maximum value within a fixed-size rectangular neighbourhood region as shown in Figure 2.3. The usage of pooling is motivated by the fact that it makes the representation approximately invariant to small translations of the input. This means that if we translate the input by a small amount, the value of most of the pooled outputs do not change.

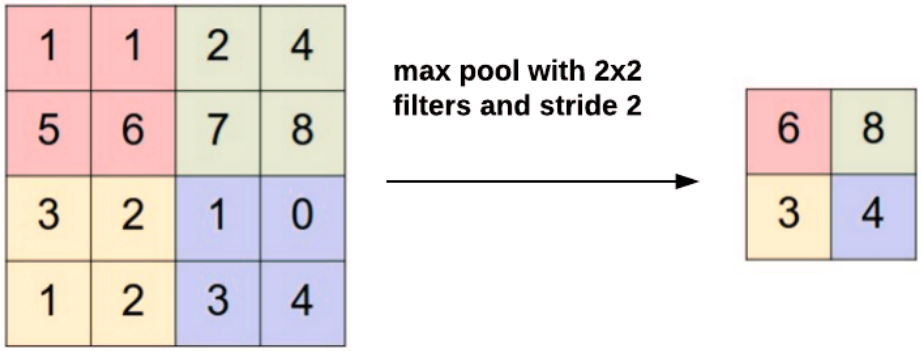


Figure 2.3: The most common downsampling operation max pool, shown with a stride of 2. That is, each max is taken over 4 numbers.

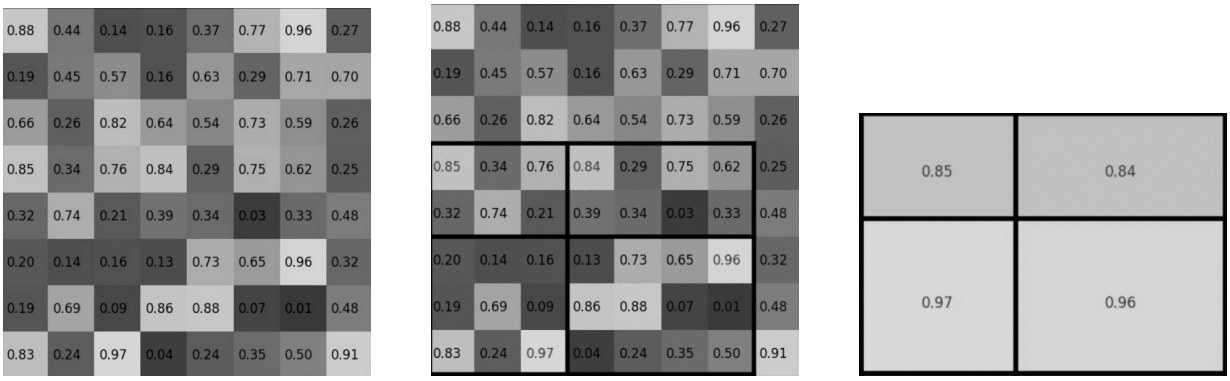
2.2.5 Region of Interest(RoI) Pooling

Region of interest pooling (also known as RoI pooling) is an operation widely used in object detection tasks using convolutional neural networks. This pooling technique was proposed in Fast R-CNN [25]. The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent.

ROI pooling employs the following steps to transform the input:

- Divide the region proposal into fixed-sized sections where the number of these sections matches the output size.
- Find the largest value in each section.
- Copy these max values to the output buffer.

Figure 2.4 shows the concept of RoI pooling visually. The RoI operation allows us to reuse the feature map from the convolutional network and hence it significantly speeds up the network processing speed. This is because if there are many object proposals in the frame, the same input feature map can be reused for all of them. And since computing the convolutions is very expensive, this approach saves significant amount of computation time. In addition, it helps to train an RPN end-to-end to generate high-quality region proposals which are then used by the second stage network for detection.



(a) Input image.

(b) Proposed region divided into sections.

(c) Max-pooled output.

Figure 2.4: ROI pooling example visualization. In this example, the RoI is divided by 2 along each dimension, where the sections are unequal due to uneven dimensions.

2.3 Object Detection

Object detection is a core problem in computer vision. Object detection is achieved through solving two tasks, object classification and localization. Object classification is the task of specifying which of k object categories the input belongs to. To solve the classification task, the learning algorithm is asked to produce a function $f : \mathbb{R}^n \rightarrow 1, ..., k$. When $y = f(x)$, the model assigns an input described by vector x to a category identified by numeric code y. For the task of object classification the input is usually an image, where an image is described as a set of pixel brightness values, and the output is a numeric code identifying the type of object in the image. Regression on the other hand returns a numeric value, as opposed to a numeric code representing a class. To solve this task, the learning algorithm is asked to output a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. This is similar to classification, except that the format of output is different [29]. Regression for the task of object detection requires returning numbers indicating the location of the detected object. For instance in the case of 2D object detection in image space, the algorithm provides 4 numbers such as $(x_1, y_1, width, height)$ representing a bounding box in image space. Image segmentation extends the task of locating the objects by identifying the object boundaries such as lines and curvature in the image. Different object detection tasks are visualized in Figure 2.5.

Since the actual goal of an object detector model is to generate exactly one detection per object, i.e. exactly one high confidence detection, a common practice is to assume that highly overlapping detections belong to the same object and collapse them into one

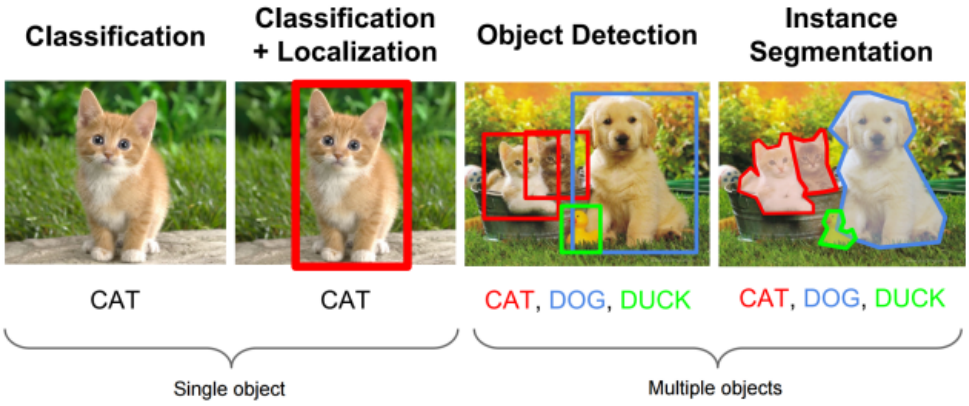


Figure 2.5: Example of single and multi-instance object classification, object classification + localization and object segmentation [12].

detection. Non-maximum suppression (NMS), is a post-processing algorithm responsible for merging all detections that belong to the same object. The algorithm greedily selects high scoring detections and deletes close-by less confident neighbors since they are likely to cover the same object by sorting the boxes based on scores.

Detection pipelines generally start by extracting a set of robust features from input images (Haar [51], SIFT [49], HOG [8], convolutional features [11]). These features are then used to identify objects using techniques discussed in more detail in the next section.

2.3.1 Classic Object Detectors

Object detection using the sliding-window technique has a long and rich history. The sliding-window technique works by sliding a box around image and classifying each image crop inside a box. In other words, the classifier is run at evenly spaced locations over the entire image, focusing on smaller areas of the feature map. One of the earliest successes of object classification is the classic work of LeCun *et al.* who applied convolutional neural networks to handwritten digit recognition [71, 42]. The introduction of Histograms of Oriented Gradients (HOG) [8] and integral channel features [10] gave rise to effective methods for object detection.

The HOG detector in particular applies a sliding window to find objects in 4 steps:

1. Scan the image by sliding the window at all locations

2. Extract features in window

3. Classify and accept window if score is above threshold

4. Clean up the overlapping bounding boxes (post-processing by appling NMS.)

Typically the window is fixed-sized. However in a typical scene, same object category may appear at different sizes and scales. To solve this issue, the authors in [8] proposed to shrink the image and repeat the sliding over many iterations at different scales. This resulted in a full image pyramid where the detector slides at every scale. The input image is first divided into non-overlapping cells, typically 8×8 pixels. The authors then proposed to apply HOG to extract features in each cell. Then a linear SVM is used to predict the presence or absence of object class in each window in the image. Finally using NMS algorithm at each iteration the highest scoring boxes are selected as the final detection. The HOG detector models an object class as a single rigid template as shown in Figure 2.6.

Figure 2.6: Single HOG template models people in upright pose [8].

One limitation of HOG was the fact that it did not take into account outlier objects such as objects in unusual poses, and instead it used a restrictive template for detecting people. The method for generating such templates assume common seen objects such as people in upright pose. Deformable Parts Model (DPMs) [21] employs a disjoint pipeline to extract static features, classify regions and then predicting bounding boxes for high scoring regions. This approach takes into account the fact that objects are composed of parts at

specific relative location. Different instances of the same object class also have parts in slightly different locations. The DPM model borrowed the idea of the HOG detector, taking a HOG template for the full object. Given an object template the DPM algorithm then adds locations relative to each other, where relative in this context means that if an object moves, the parts will also move along together. This idea is visualized in Figure 2.7, where an object is decomposed into parts. In addition, DPM gives some slack to the location of the part, i.e. a part can also have deformation, meaning that it can slightly move around the expected location. Such assumption helps in terms of detecting people with different heights and sizes and not just an average person.

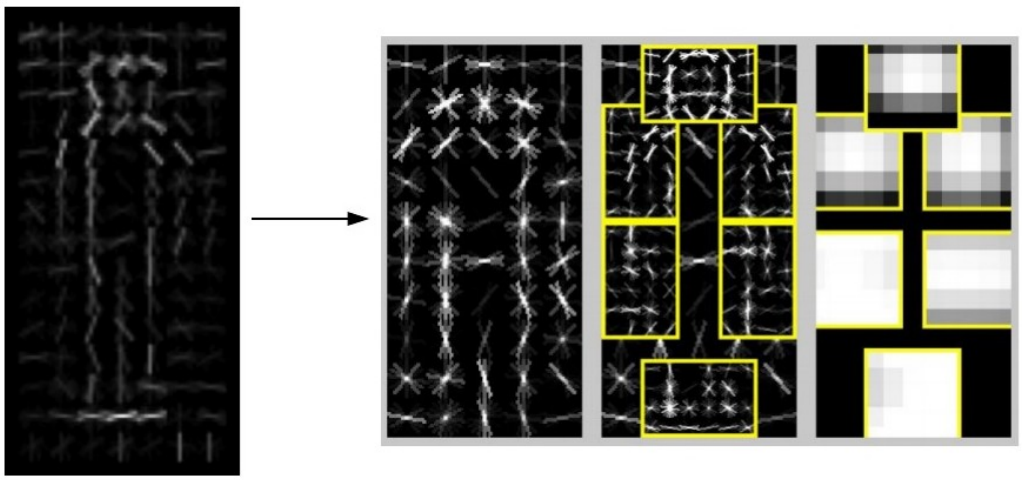


Figure 2.7: Example of DPM full detector model with the added parts [21].

Note that as shown in Figure 2.7 each part has an **appearance**, which is modeled with a HOG template. DPMs significantly helped to extend dense detectors to more general object categories and had top results on PASCAL [20] for many years. Using sliding window techniques such as HOG and DPMs to perform object detection has limitations in terms of computation time and accuracy. Many viewing windows may contain a perfectly identifiable portion of the object, for instance part of an object, but not the entire object, nor even the center of the object. This leads to decent classification but poor localization in general. While the combination of hand-crafted features and sliding-window approach was the leading detection paradigm in classic computer vision, with the emergence of

deep learning [40], two-stage detectors, described next, quickly came to dominate object detection.

2.3.2 Two-Staged Detectors

The dominant paradigm in modern object detection takes a different approach from the sliding-window technique by adopting a two-stage detection. As pioneered in the Selective Search work [70], the first stage generates potential bounding boxes that should contain all the objects, and the second stage classifies the proposals into foreground and background classes. The overall pipeline for Selective Search includes generating a set of candidate proposals, extracting features using a convolutional network, scoring the boxes using an SVM model, adjusting the bounding boxes using a linear model, and finally eliminating duplicate detections using NMS. Each stage of this complex pipeline requires precise tuning and the resulting system is slow, taking more than 40 seconds per image at test time. Regions with CNN features (R-CNN) [26] upgraded the second-stage classifier to a convolutional network providing a higher gain in accuracy. R-CNN and its variants use region proposals instead of sliding windows to find objects in images. R-CNN was improved over the years, both in terms of speed [32] and by using learned object proposals [52, 60], which enable training the systems end-to-end. Spatial Pyramid Pooling (SPPNet) [32] speeds up the original R-CNN approach. It introduces a spatial pyramid pooling layer that is more robust to region size and scale, allowing the classification layers to reuse features computed over feature maps generated at different image resolutions. Fast R-CNN [25] builds on R-CNN and SPPNet by speeding up the network's convolution processing time by sharing computation. Fast R-CNN proposed an architecture to fine-tune all layers end-to-end by minimizing a loss for both confidence scores and bounding box regression, which was first introduced in MultiBox [69] for learning objectness. A Fast R-CNN network takes an entire image as input and a set of object proposals. The network first processes the whole image with several convolutions and max pooling layers to produce a feature map. Then for each object proposal, a region of interest (RoI) pooling layer was proposed to extract a fixed-length feature vector from the feature map. Each feature vector is then fed into a sequence of FC layers that finally branch into two output layers: one that produces K object classes and another layer that outputs four real-valued numbers, representing the bounding-box position of the detected object. The architecture of Fast R-CNN is depicted in Figure 2.8.

Faster R-CNN took the idea of optimizing the computation time further by introducing a Region Proposal Network (RPN) that shares full-image convolutional features with the second stage detection network. Faster R-CNN introduces a method to integrate the

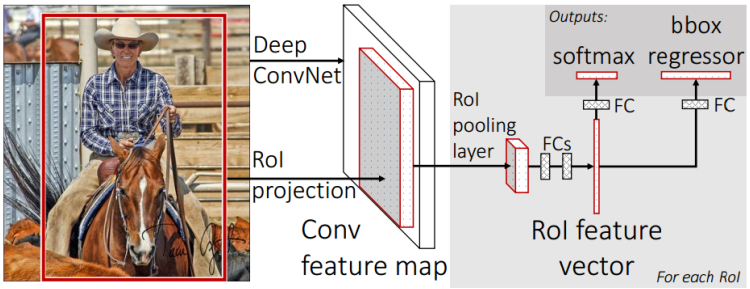


Figure 2.8: Fast R-CNN as a two-staged detector architectural diagram [25].

RPN with Fast R-CNN by alternating between fine-tuning shared convolutional layers and prediction layers for these two networks. This work also introduced a training scheme that alternates between fine-tuning for the object proposal task and then fine-tuning for object detection. This training scheme allows the network to converge quickly producing a unified network with convolution features shared by both tasks [60]. The architecture of Faster R-CNN is depicted in Figure 2.9, which shows how the RPN shares the convolutional features corresponding to the full image with the second stage detection network, thus saving computation time at each detection stage. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection.

2.3.3 Single-Stage Detectors

There have been many attempts to build faster detectors by optimizing each stage of the detection pipeline, but so far, significantly increased speed comes only at the cost of significantly decreased detection accuracy.

As an alternative approach to speed up the 2D detectors, OverFeat [63] was one of the first single stage object detector using a multi-scale and sliding window approach with ConvNets. OverFeat predicts a bounding box directly from each location of the feature map by accumulating predicted bounding boxes. They argue that combining many localization predictions removes the need to train on background samples.

More recently Single Shot Detector (SSD) [48, 22] and You Only Look Once (YOLO) [57, 58] have renewed interest in single stage methods. These detectors have been tuned for speed but their accuracy trails that of two-stage methods. SSD on average has around $10-20\%$ lower AP on datasets such as PASCAL VOC [18], while YOLO focuses on more

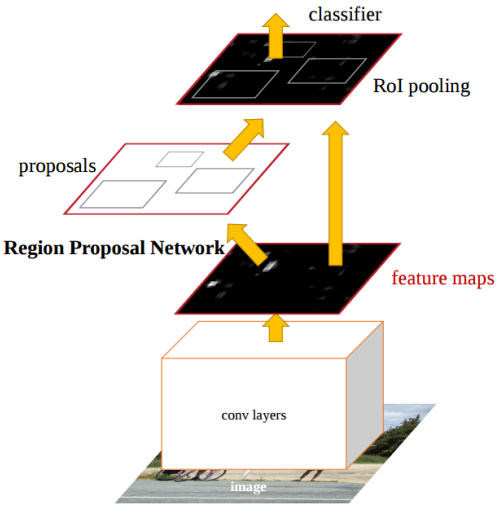


Figure 2.9: Faster R-CNN as a two-staged detector architectural diagram [60].

extreme speed/accuracy trade-off. Recent work showed that two-stage detectors can be made fast simply by reducing input image resolution or the number of proposals, but single stage methods still cannot gain the same accuracy offered by a two-stage detection system [35].

YOLO discretizes the image into an $S \times S$ grid. Then each grid cell, using convolutional features, proposes potential bounding boxes along with their confidence scores. The difference between this approach and an RPN-based approach is that YOLO uses the features from the entire image to predict each bounding box, where each individual grid cell predicts B bounding boxes with confidence score. These scores can be interpreted as whether they are likely to contain an object or not. If there is no object in a grid cell, it is expected for the confidence score to be zero. YOLO [57] shares some similarities with R-CNN. But instead of relying on the initial bounding box guesses, it puts spatial constraints on the grid cell proposals, which helps to mitigate multiple detections of the same object. YOLO uses the whole feature map, as shown in Figure 2.10, to predict confidences for multiple categories and bounding boxes.

SSD discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. SSD is similar to the RPN in Faster R-CNN since they use a fixed set of anchor boxes for prediction. However instead of pooling features from these anchors and then evaluating them separately via a second clas-

sifier, they simultaneously produce a score for each object category in each box. Therefore with SSD there is no proposal step. In addition, the network combines predictions from multiple feature maps with different resolutions to handle objects of various sizes [48].

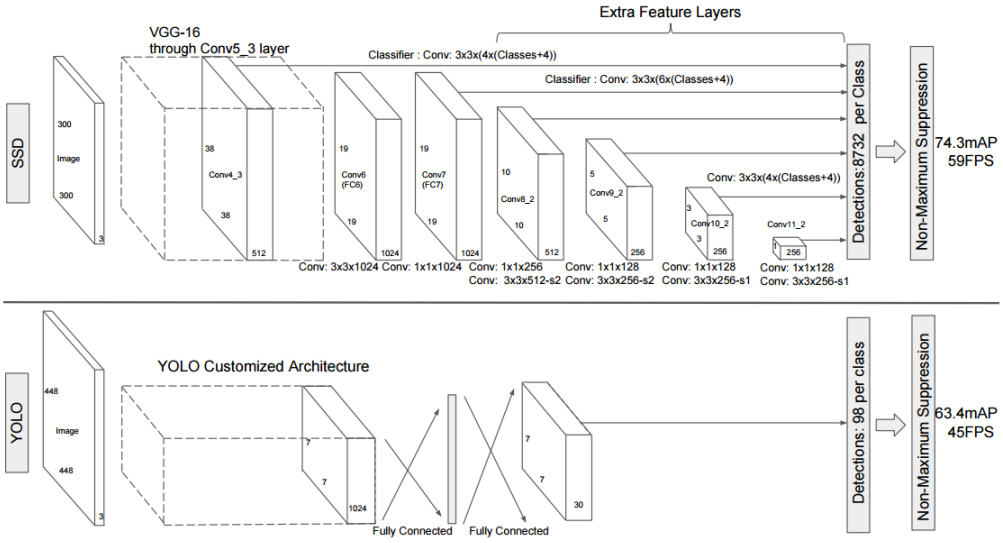


Figure 2.10: Comparison between two single stage detection models: SSD and YOLO [48]. This figure highlights the main difference between SSD and YOLO in terms of adopting different feature extraction mechanism. That is, YOLO uses the final feature map but SSD uses features at different scales.

2.4 3D Object Detection

Following the trend in 2D two-stage detection, 3D proposal generation algorithms typically used hand-crafted features to generate a small set of candidate boxes that retrieve most of the objects in 3D space. 3DOP [4] uses hand-crafted geometric features from stereo point clouds to score 3D sliding windows in an energy minimization framework. The top K scoring windows are selected as region proposals, which are then consumed by a modified Fast-RCNN [25] to generate the final 3D detections. Mono3D [3] uses the same framework, but instead exploits a ground plane prior and hand-crafted features from semantic segmentation outputs to generate 3D proposals from monocular images.

Single stage or single shot object detectors similar to YOLO [58] have also been proposed as RPN free architectures for the 3D object detection task. VeloFCN [44] projects a LIDAR point cloud to the front view, which is used as an input to a fully convolutional neural network to directly generate dense 3D bounding boxes. 3D-FCN [43] extends this concept by applying 3D convolutions on 3D voxel grids constructed from LIDAR point clouds to generate better 3D bounding boxes.

Another alternative to 3D RPN-based architectures is 3D object detection from monocular images only. Deep MANTA [2] proposes a many-task vehicle analysis approach from monocular images that optimizes region proposal, detection, 2D box regression, part localization, part visibility, and 3D template prediction simultaneously. The architecture requires a database of 3D models corresponding to several types of vehicles, making the proposed approach hard to generalize to classes where such models do not exist. Deep3DBox [50] proposes to extend a 2D object detector [1] to 3D by exploiting the fact that the perspective projection of a 3D bounding box should fit tightly within its 2D detection window. However, the performance of these methods, which is presented in Section 3.3, is consistently weaker than methods that use point cloud data.

Recently, VoxelNet [76] was released as a single shot detector that leverages more powerful point-wise features extracted directly on a 3D voxel grid. However, even with sparse 3D convolution operations, the inference time for the VoxelNet is $225ms$ on a TitanX GPU [76].

3D RPNs have previously been proposed in [67] for 3D object detection from RGBD images, where MV3D extends the image based RPN of Faster R-CNN [60] to 3D by corresponding every pixel in the *BEV* feature map to multiple prior 3D anchors. These anchors are then fed to the RPN to generate 3D proposals that are used to create view-specific feature crops from the BEV, front view of [44], and image view feature maps. A *deep fusion* scheme is used to combine information from these feature crops to produce the final

detection output.

One substitute for RPNs present in the literature is using mature 2D object detectors for proposal generation in 2D, followed by amodal 3D extent regression. This trend started with [41] for indoor object detection, which inspired Frustum-based PointNets (F-PointNet) [54] to extend the feature extraction stage to rely on point-wise features of PointNet [55] instead of point histograms. Similarly PointFusion [74] proposed applying a 2D detector to obtain bounding boxes, and then fusing corresponding image and 3D point cloud to perform object detection. They process image data and the raw point cloud data independently by a CNN and a PointNet architecture, respectively. While these methods work well for brightly lit scenes, they are expected to perform poorly in more extreme scenarios since they are designed to rely on the image and the quality of the 2D detector. Such extreme scenarios can be caused by sensor noise and severe lighting variations, such as night time or severe weather conditions such as snow. These conditions are typically present in autonomous driving scenarios.

2.5 Evaluation Metrics for Object Detection

This section discusses the evaluation metrics used to evaluate AVOD, both at RPN stage and the overall performance of the second stage classification, regression and heading estimation.

2.5.1 Bounding Box Evaluation

For a bounding box to be considered a correct detection, the area of overlap a_o between the predicted bounding box B_p and ground truth bounding box B_{gt} must exceed a certain threshold. A common threshold for 2D is 0.7 (70%), obtained by the formula :

$$a_o = \frac{area \quad B_p \cap B_{gt}}{area \quad B_p \cup B_{gt}} \tag{2.4}$$

where $B_p \cap B_{gt}$ denotes the intersection of the predicted and ground truth bounding boxes and $B_p \cup B_{gt}$ their union [19]. This formula can be extended to 3D to measure how well the 3D boxes overlap.

2.5.2 Precision & Recall

For a given task and class, the precision-recall curve is computed from a method's ranked output. Recall is defined as the proportion of all positive examples ranked above a given rank. Precision is the proportion of all examples above the rank which are from the positive class [19].

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \tag{2.5}$$

where TP is acronym for True Positive, which indicates a correct detection with existing corresponding ground-truth. FP is False Positive and indicates that an object was detected but it does not have a corresponding ground-truth, i.e. false detection. FN is False Negative, and indicates that an object was in the ground-truth but not detected by the system.

2.5.3 Average Precision (AP)

The AP summarizes the shape of the precision-recall curve and is defined as the mean precision at a set of eleven equally spaced recall levels $[0, 0.1, ..., 1]$:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, ..., 1\}} p_{interp}(r) \tag{2.6}$$

The precision at each recall level r is interpolated by taking the maximum precision measured for which the corresponding recall exceeds r:

$$p_{interp}(r) = \max_{\hat{r}:\hat{r} \geq r} p(\hat{r}) \tag{2.7}$$

where $p(\hat{r})$ is the measured precision at recall $\hat{r}$. The intention in interpolating the precision-recall curve is to reduce the impact of the "wiggle" in the precision-recall curve caused by small variations in the ranking of examples [19]. The detections are iteratively assigned to ground truth labels starting with the largest overlap, measured by bounding box intersection over union. In order to obtain a high score, a detection method must have precision at all levels of recall. Therefore this metric penalizes methods which retrieve only a subset of detections with high precision.

2.5.4 Average Orientation Similarity (AOS)

KITTI assesses the performance of jointly detecting objects and estimating their 3D orientation using an average orientation similarity (AOS).

$$AOS = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} \max_{\hat{r}: \hat{r} \geq r} s(\hat{r}) \tag{2.8}$$

where r is recall.

The orientation similarity $s \in [0, 1]$ at recall r is a normalized ([0..1]) variant of the cosine similarity defined as:

$$s(r) = \frac{1}{|D(r)|} \sum_{i \in D(r)} \frac{1 + cos\Delta_{\theta}^{(i)}}{2} \delta_i \tag{2.9}$$

where $D(r)$ denotes the set of all object detections at recall rate r and $\Delta_{\theta}^{(i)}$ is the difference in angle between estimated and ground truth orientation of detection i. To penalize multiple detections which correspond to a single object, we set $\delta_i = 1$ if detection i has been assigned to a ground truth bounding box (overlaps by at least 50%) and $\delta_i = 0$ if it has not been assigned.

Finally KITTI evaluates pure classification and regression (continuous orientation) performance on the task of 3D object orientation estimation in terms of orientation similarity [23].

Chapter 3

AVOD

This chapter discusses AVOD, an **A**ggregate **V**iew **O**bject **D**etection architecture for autonomous driving.

3.1 The AVOD Architecture

The AVOD architecture is depicted in Figure 3.1, where it uses feature extractors to generate feature maps from both the BEV map and the RGB image. Both feature maps are used by the RPN to generate non-oriented region proposals, which are passed to the detection network for dimension refinement, orientation estimation, and category classification.

3.1.1 RGB Image Input

The RGB input image is expected to provide better classification and size information in comparison to LIDAR point cloud data. KITTI dataset image data was collected using a two color and two grayscale PointGrey Flea2 video cameras (10 Hz, resolution: 1392×512 pixels, opening: $90^\circ \times 35^\circ$)[23]. KITTI provides the data for training and testing using the left color camera images. To preserve features of smaller objects, input images are resized to 1590×480 pixels. However the size of the input image can also be set as a hyperparameter. Input feature normalization is done by subtracting the mean of the RGB channels over the training set, for each image.

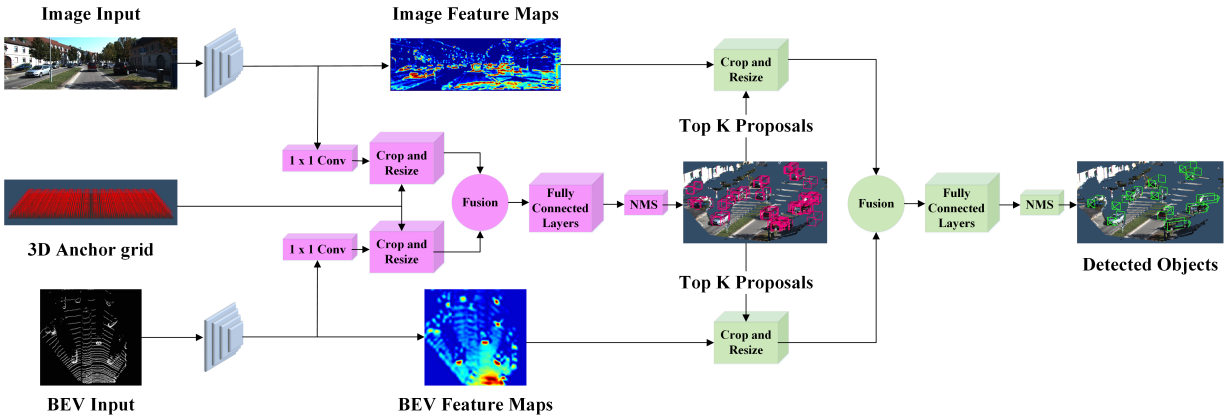


Figure 3.1: AVOD's architectural diagram. The feature extractors are shown in **blue**, the region proposal network in **pink**, and the second stage detection network in **green**.

3.1.2 Point Cloud Representation

KITTI dataset LIDAR was collected using a Velodyne HDL-64E 3D laser scanner (10 Hz, 64 laser beams, range: 100 m)[23]. The 3D point cloud of the road scene is used to carry out a more accurate 3D object detection and localization. A raw point cloud is not useful for convolutional neural networks, because a 3D grid representation of the point cloud requires complex and extensive computation for feature extraction. Similar to [6] we can encode the information from the point cloud into useful compact 2D representation. This enables us to leverage powerful existing feature extraction networks such as VGG, which work on 2D image inputs.

Bird's Eye View Maps

The bird's eye view maps encode maximum point cloud height and density information. Compared to the RGB image input, this view offers several advantages. The physical sizes of objects are preserved when projected into the bird's eye view. Furthermore, there are no issues with with overlapping occlusion since objects in this view occupy different space. The bird's eye view maps are generated by discretizing the point cloud with a 0.1 meters resolution and then projecting the voxels onto the xz-plane. For each cell, the height feature is computed as the maximum height of the points in the cell. In order to encode more detailed height information, the point cloud is divided into M equally-sized slices. A height

map is computed for each slice. The point cloud density indicates the number of points in each cell. This results to a six-channel BEV map from a voxel grid representation of the point cloud at 0.1 meter resolution. The point cloud is cropped at $[-40, 40] \times [0, 70]$ meters, along the x and z-axis respectively, to contain points within the field of view of the camera. The first 5 channels of the BEV map are encoded with the maximum height of points in each grid cell, generated from 5 equal slices between $[0, 2.5]$ to include points above the ground-plane. The sixth BEV channel contains point density information computed per cell, normalized by computing the $\min(1.0, \frac{\log(N+1)}{\log 16})$, where N is the number of points in the cell. It must be noted that this method takes certain assumptions such as an accurate estimated ground-plane. Another assumption is that objects such as cars and pedestrian appear within certain distance above the ground-plane. The final input to the network is a BEV map of size 800×700 with information encoded into 6 channels.

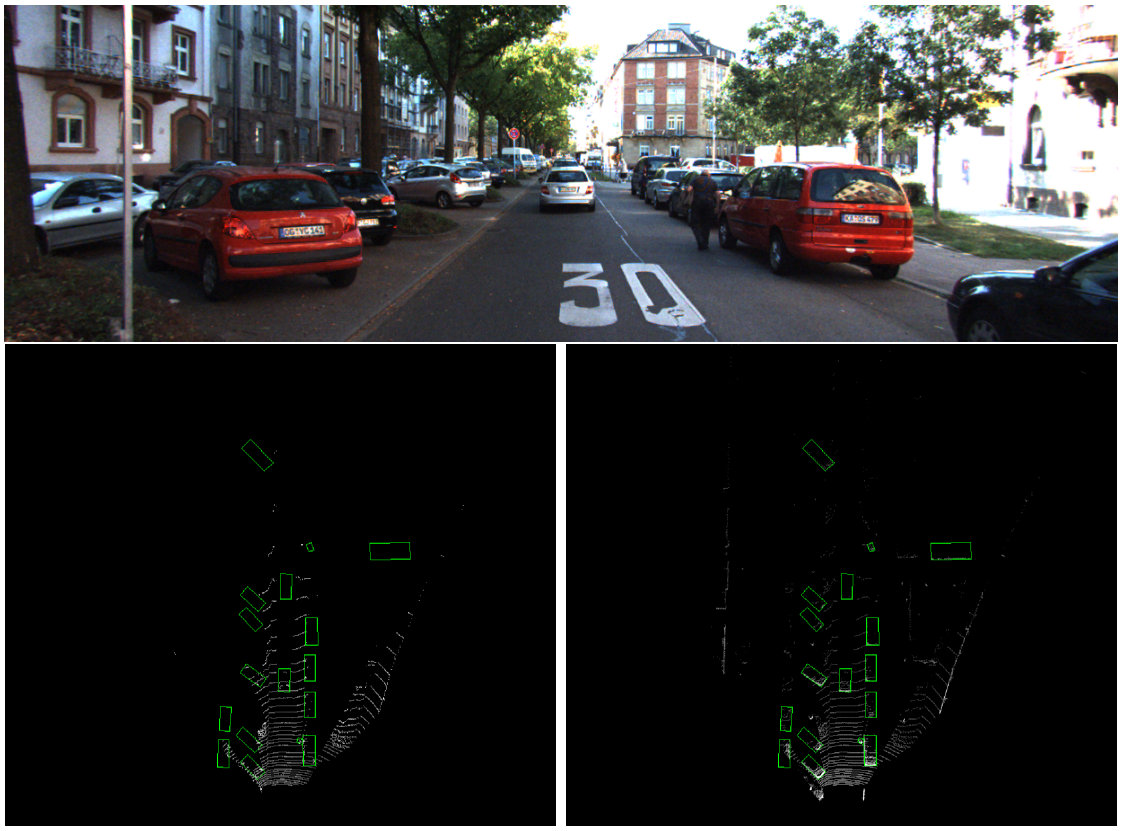


Figure 3.2: **Bird's Eye View Maps**, top representing the 2D image, and bottom the bird's eye view maps from LIDAR point clouds showing 1 height slice and density map.

3.1.3 Generating Feature Maps From Point Clouds And Images

VGG Feature Extractor

AVOD uses two feature extractors, one for each input view. The feature extractors are based on the VGG16 architecture [65], with the following modifications. First, half the number of filters are used at every convolutional layer. Second, Xavier weight initialization method [27] is used to initialize weights instead of using pre-trained ImageNet weights. Third, batch normalization is employed to provide similarly scaled feature maps from both views. Finally, the fourth maxpooling layer and fifth convolutional layers of VGG16 are discarded, resulting in output feature maps that have 8× smaller resolution than their corresponding input, with a depth of 256. The output of the feature extraction from both views is passed through a 4× bilinear upsampling layer to attain higher resolution feature maps. A visual representation of the extracted features can be seen in Figure 3.3.

Figure 3.3: Visual representation of a subset of image features tiled at *Conv3* layer.

Number of Layers	Operation	Kernel	Output
2	conv1	3×3	$480 \times 1590 \times 32$
1	maxpool	2×2	$240 \times 795 \times 32$
2	conv2	3×3	$240 \times 795 \times 64$
1	maxpool	2×2	$120 \times 397 \times 64$
3	conv3	3×3	$120 \times 397 \times 128$
1	maxpool	2×2	$60 \times 198 \times 128$
3	conv4	3×3	$60 \times 198 \times 256$
NA	upsampling	*NA*	$240 \times 795 \times 256$
1	1×1 conv	1×1	$240 \times 795 \times 1$

Table 3.1: Image branch feature extractor layers with image input of ($480 \times 1590 \times 3$).

Number of Layers	Operation	Kernel	Output
2	conv1	3×3	$700 \times 800 \times 32$
1	maxpool	2×2	$350 \times 400 \times 32$
2	conv2	3×3	$350 \times 400 \times 64$
1	maxpool	2×2	$175 \times 200 \times 64$
3	conv3	3×3	$175 \times 200 \times 128$
1	maxpool	2×2	$87 \times 100 \times 128$
3	conv4	3×3	$87 \times 100 \times 256$
NA	upsampling	*NA*	$350 \times 400 \times 256$
1	1×1 conv	1×1	$350 \times 400 \times 1$

Table 3.2: BEV branch feature extractor layers with BEV input of ($700 \times 800 \times 6$).

Feature Pyramids

Inspired by the Feature Pyramid Network (FPN) [46], we design a bottom-up decoder that learns to upsample the feature map back to the original input size. The decoder takes as input the output of the encoder, F, and produces a new $M \times N \times \tilde{D}$ feature map. Figure 3.4 shows the operations performed by the decoder, which include upsampling of the input via a *conv-transpose* operation, concatenation of a corresponding feature map from the encoder, and finally fusing the two via a 3×3 convolution operation. The final feature map is of high resolution *and* representational power, and is shared by both the RPN and the second stage detection network. The combination of features at different scales can significantly boost the performance of the network for detecting small objects such as pedestrian and cyclists. The performance boost of applying feature pyramids is further discussed in Section 3.3.

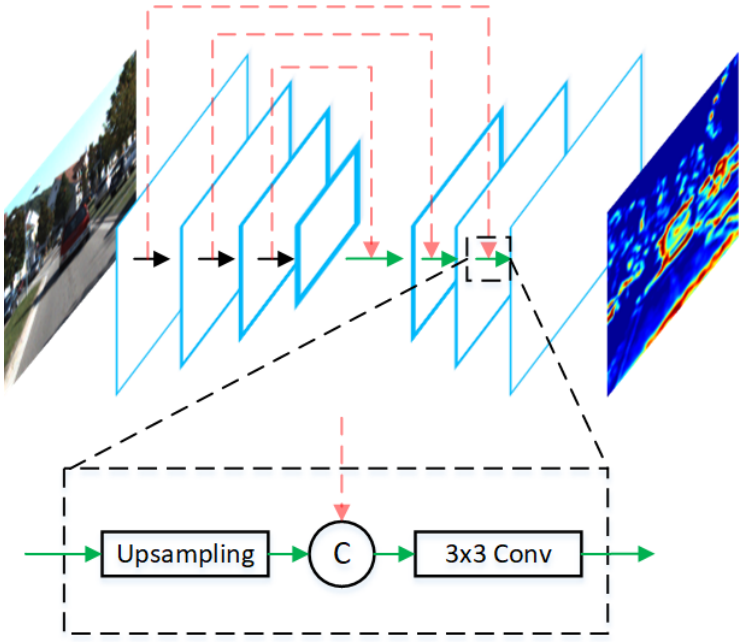


Figure 3.4: The architecture of the feature pyramid extractor shown here for the image branch. Feature maps are propagated from the encoder to the decoder section via red arrows. Fusion is then performed at every stage of the decoder by a learned upsampling layer, followed by concatenation (represented as C), and then mixing via a convolutional layer, resulting in a full resolution feature map at the last layer of the decoder.

Number of Layers	**Operation**	**Kernel**	**Output**
2	conv1	3×3	$360 \times 1200 \times 32$
1	maxpool	2×2	$180 \times 600 \times 32$
2	conv2	3×3	$180 \times 600 \times 64$
1	maxpool	2×2	$90 \times 300 \times 64$
3	conv3	3×3	$90 \times 300 \times 128$
1	maxpool	2×2	$45 \times 150 \times 128$
3	conv4	3×3	$45 \times 150 \times 256$
1	upsample_conv3	3×3	$90 \times 300 \times 256$
1	pyramid_fusion3	3×3	$90 \times 300 \times 64$
1	upsample_conv2	3×3	$180 \times 600 \times 128$
1	pyramid_fusion2	3×3	$180 \times 600 \times 32$
1	upsample_conv1	3×3	$360 \times 1200 \times 64$
1	pyramid_fusion1	3×3	$360 \times 1200 \times 32$

Table 3.3: Image branch feature pyramid feature extractor layers with image input of $(360 \times 1200 \times 3)$. Note that the output dimension from the operation *upsample_conv* involves both the upsampling and concatenation operations.

Number of Layers	Operation	Kernel	Output
2	conv1	3×3	$704 \times 800 \times 32$
1	maxpool	2×2	$352 \times 400 \times 32$
2	conv2	3×3	$352 \times 400 \times 64$
1	maxpool	2×2	$176 \times 200 \times 64$
3	conv3	3×3	$176 \times 200 \times 128$
1	maxpool	2×2	$88 \times 100 \times 128$
3	conv4	3×3	$88 \times 100 \times 256$
1	upsample_conv3	3×3	$176 \times 200 \times 256$
1	pyramid_fusion3	3×3	$176 \times 200 \times 64$
1	upsample_conv2	3×3	$352 \times 400 \times 128$
1	pyramid_fusion2	3×3	$352 \times 400 \times 32$
1	upsample_conv1	3×3	$704 \times 800 \times 64$
1	pyramid_fusion1	3×3	$704 \times 800 \times 32$

Table 3.4: BEV branch feature pyramid feature extractor layers with BEV input of ($700 \times 800 \times 6$) with padding to allow even divisions for max pooling ops. The padding is removed from the final feature map, resulting to a final output size of ($700 \times 800 \times 32$).

3.1.4 Multimodal Fusion Region Proposal Network

Anchor Generation

Similar to 2D two-stage detectors, the RPN network regresses the difference between a set of prior 3D boxes and the ground truth. These prior boxes are referred to as anchors, where the concept of anchor is the same as proposed in Faster R-CNN.

Anchors in 3D are encoded using the *axis-aligned* bounding box encoding shown in Figure 3.5. Anchor boxes are parameterized by the centroid (t_x, t_y, t_z) and axis aligned dimensions (d_x, d_y, d_z). To generate the 3D anchor grid, (t_x, t_y) pairs are sampled at an interval of 0.5 meters in BEV, while t_z is determined based on the sensor's height above the

ground plane. Similar to [5], the dimensions of the anchors are determined by clustering the training samples for each class. Using the ground truth label clusters calculated for BEV, N 3D anchor boxes are generated. The reason for using clusters is because, it is difficult to hand pick the best prior box dimensions. The network can still learn to adjust the boxes appropriately, however if we pick better priors for the network to start with, we can make it easier for the network to learn to predict more accurate bounding box detections. Therefore instead of choosing priors by hand, the ground truth label sizes from the training set are clustered using k-means into 2, 1, and 1 clusters, for the Car, Pedestrian and Cyclist classes, respectively. The anchor boxes are generated on a grid over the 3D space with a stride of 0.5 meters, placing them on the ground plane. The size of the anchor stride is a hyperparameter which can be tuned to obtain the best proposals covering all the region with potential object targets on the grid. The boxes are generated with 0° and 90° orientations, resulting to 4 boxes when the cluster is 2, where the orientation is fixed during the RPN stage and orientation regression is delayed until the second stage. Since a LIDAR point cloud is sparse, this results in many empty anchors. Therefore anchors without 3D points in BEV are removed efficiently by computing an integral image over the point occupancy map, resulting in $10 - 100$K non-empty anchors per frame, depending on the sample and the anchor stride.

Multiview Feature Crop And Resize Operation

To extract feature crops for every anchor from the view specific feature maps, we use the crop and resize operation [35]. Given an anchor in 3D, two regions of interest are obtained by projecting the anchor onto the BEV and image feature maps. The corresponding regions are then used to extract feature map crops from each view, which are then bilinearly resized to 3×3 to obtain equal-length feature vectors.

Dimensionality Reduction Via 1×1 Convolutional Layers

In some scenarios, the region proposal network is required to save feature crops for 100K anchors in GPU memory. Attempting to extract feature crops directly from high dimensional feature maps imposes a large memory overhead per input view. As an example, extracting 7×7 feature crops for 100K anchors from a 256-dimensional feature map requires around 5 gigabytes[1] of memory assuming 32-bit floating point representation. Furthermore, processing such high-dimensional feature crops with the RPN greatly increases its computational requirements.

[1] $100,000 \times 7 \times 7 \times 256 \times 4$ bytes.

Inspired by their use in [37], we apply a 1×1 convolutional kernel on the output feature maps from each view, as an efficient dimensionality reduction mechanism. The 1×1 convolution acts on every pixel position in each feature map according to:

$$f_{out} = \sigma(\sum_{i=0}^{256} w_i f_i + b), \tag{3.1}$$

where f_i is the pixel value at each one of the 256 feature map channels, w_i is a learned weight, and b is a learned bias term. These 1×1 convolutions can be thought of as strictly linear coordinate-dependent transformations in the filter space, followed by a non-linear activation function σ. By learning these transformations, the RPN reduces the dimensionality of its input feature maps while retaining information deemed useful for the proposal generation task. From a computational point of view, 1×1 convolutions are dot products across the depth of the feature map and can be computed efficiently on GPUs. The result is a feature selection mechanism that reduces the memory overhead for computing anchor specific feature crops by $256\times$ per view, allowing the RPN to process fused features of tens of thousands of anchors using only a few megabytes of additional memory while still attaining state-of-the-art performance.

3D Proposal Generation

The outputs of the crop and resize operation are equal-sized feature crops from both views, which are fused via an element-wise mean operation. Two task specific branches [60] composed of 2 fully connected layers of size 256, use the fused feature crops to regress axis aligned object proposal boxes and output an object/background "objectness" score. 3D box regression is performed by computing $(\Delta t_x, \Delta t_y, \Delta t_z, \Delta d_x, \Delta d_y, \Delta d_z)$, the difference in centroid and dimensions between anchors and ground truth bounding boxes.

Loss Function

Smooth L1 loss is used for 3D box regression, and cross-entropy loss for "objectness". Similar to [60], background anchors are ignored when computing the regression loss. Background anchors are determined by calculating the 2D IoU in BEV between the anchors and the ground truth bounding boxes. For the *car* class, anchors with IoU less than 0.3 are considered background anchors, while ones with IoU greater than 0.5 are considered object anchors. This way, a single ground-truth box may assign positive labels to multiple anchors. Anchors that are neither positive or negative do not contribute to the training

objective. Note that this is to eliminate any ambiguity between a positive and negative sample. For the *pedestrian* and *cyclist* classes, the object anchor IoU threshold is reduced to 0.45. To remove redundant proposals, 2D NMS at an IoU threshold of 0.8 is used to keep the top 1024 proposals during training. For NMS, tensorflow API is used which selects a subset of bounding boxes based on scores, by pruning away boxes that have high IoU overlap with previously selected boxes. At inference time, 300 proposals are used for the Car class, whereas 1024 proposals are kept for pedestrians and cyclists. These values are deduced from the recall vs. number of proposals curves (Figure 3.9) to obtain the highest possible recall for every class with the least number of proposals.

We minimize an objective function following the multi-task loss in Fast R-CNN [25].

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{3.2}$$

In the above loss formulation, i is the index of an anchor in a mini-batch and p_i is the predicted probability of anchor i being an object. The ground-truth label p_i^* is 1 if the anchor is positive and is 0 if the anchor is negative. t_i is a vector representing the 6 parameterized coordinates of the predicted bounding box, and t_i^* is the ground-truth box associated with a positive anchor. The classification loss L_{cls} is log loss over two classes (object vs. non-object). For the regression loss, we use $L_{reg}(t_i, t_i^*) = R(t_i, t_i^*)$ where R is the robust loss function as define in Eq. 3.2. The term $p_i^* L_{reg}$ means the regression loss is activated only for positive anchors ($p_i^* = 1$) and is disabled otherwise ($p_i^* = 0$). The output of the cls and reg layers consist of p_i and t_i repectively. The two terms are normalized by N_{cls} and N_{reg} and weighted by a balancing hyperparameter λ. In the case of the cls term, it is normalized by the anchor mini-batch size and the reg term is normalized by the number of positive anchors. By default we set λ to 5 for reg which provides the best performance in our experiments.

3.1.5 Second Stage Detection Network

3D Bounding Box Encoding

In [6], Chen et al. claim that 8 corner box encoding provides better results for regressing oriented bounding boxes than the traditional axis aligned encoding previously proposed in [67]. However, an 8 corner encoding does not take into account the physical constraints of a 3D bounding box, as the top corners of the bounding box should be forced to align with those at the bottom. To reduce redundancy and keep these physical constraints, we

propose to encode the bounding box with four corners and two height values representing the top and bottom corner offsets from the ground plane, determined from the sensor height.

Figure 3.5 shows different bounding box representations. Our regression targets are therefore $(\Delta x_1, ..., \Delta x_4, \Delta y_1, ..., \Delta y_4, \Delta h_1, \Delta h_2)$, the corner and height offsets from the ground plane between the proposals and the ground truth boxes. To determine corner offsets, we correspond the closest corner of the regressed bounding boxes to the closest corner of the ground truth box in BEV. The proposed encoding reduces the box representation from 24 dimensional vector to a 10 dimensional one. To make a comparison between different bounding box representations, an 8-corner box representation was also implemented. The regression targets for 8-corner are: $t = (\Delta x_1, ..., \Delta x_8, \Delta y_1, ..., \Delta y_8, \Delta z_1, ..., \Delta z_8)$. Two variations of 8-corner are implemented, one where the order of the corners are preserved during conversion, and another formulation where the order of corners are not preserved but rather the corners are rotated to the nearest 90° angle. This enables us to perform a closest corner to corner matching when comparing the corners to the ground-truth boxes. When regressing corners, it is possible for the regressed corners to be skewed. In our implementation, we align each face corners of the box, resulting to an aligned 3D box shape. This process is visualized in Figure 3.6 for four corner encoding, where the alignment is done as follows. First the mid-points on each edge are determined, and the opposite mid-points are connected via a straight line. The longest line is then selected and all the corners are then aligned with respect to this line, by selecting the minimum and maximum corner values along each axis. Section 3.7 shows the performance comparison between these box representations.

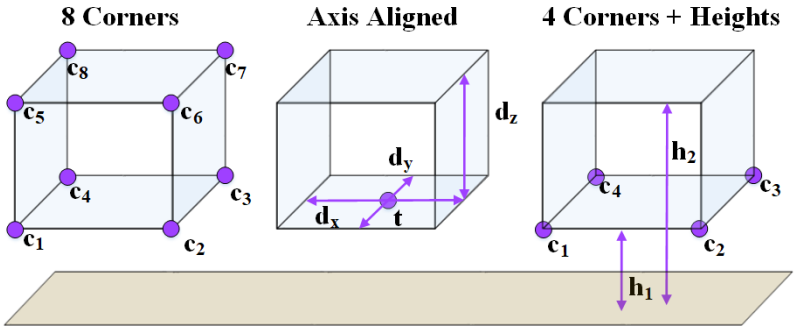


Figure 3.5: A visual comparison between the 8 corner box encoding proposed in [6], the axis aligned box encoding proposed in [67] and proposed 4 corner encoding.

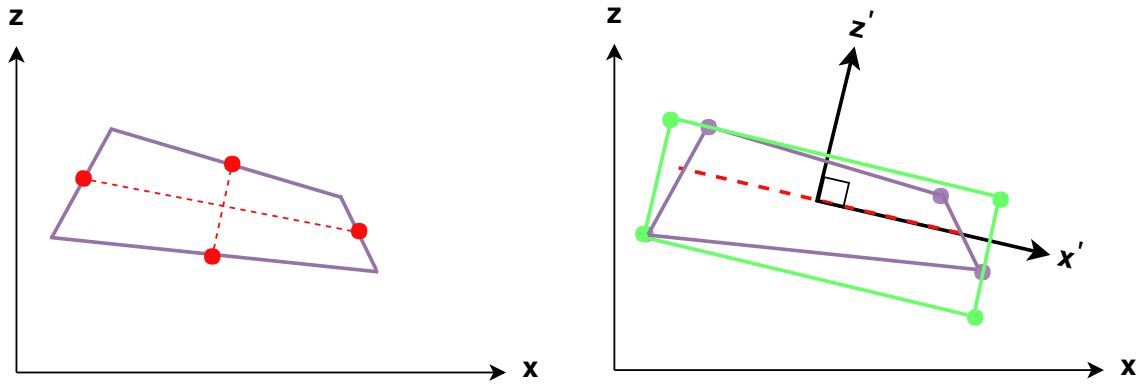


Figure 3.6: A visual example of aligning skewed regressed 4 corners. The left figure shows how the reference line is selected along the longest side. The right figure shows how the corners are aligned with respect to the selected line.

Explicit Orientation Vector Regression

To determine orientation from a 3D bounding box, MV3D [6] relies on the extents of the estimated bounding box where the orientation vector is assumed to be in the direction of the longer side of the box. This approach suffers from two problems. First, this method fails for detected objects that do not always obey the rule proposed above, such as pedestrians. Secondly, the resulting orientation is only known up to an additive constant of $\pm\pi$ radians. Orientation information is lost as the corner order is not preserved in closest corner to corner matching. Figure 3.8 presents an example of how the same rectangular bounding box can contain two instances of an object with opposite orientation vectors. Our architecture remedies this problem by computing $(x_{or}, y_{or}) = (\cos(\theta), \sin(\theta))$. This orientation vector representation implicitly handles angle wrapping as every $\theta \in [-\pi, \pi]$ can be represented by a unique unit vector in the BEV space. We use the regressed orientation vector to *correct* the bounding box orientation estimate from the adopted four corner representation. Specifically, we extract the four possible orientations of the bounding box, and then choose the one closest to the explicitly regressed orientation vector. This idea is visualized in Figure 3.7. In Section 3.3, this process is shown to provide a more accurate orientation estimate than directly using the regressed orientation vector.

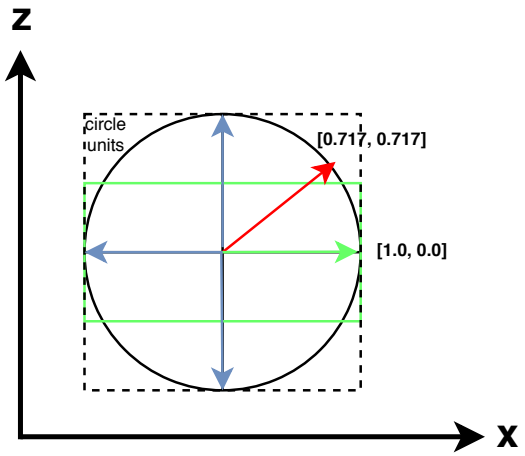


Figure 3.7: A visual example of orientation correction. The red arrow represents the regressed orientation vector. Blue and green arrows are the possible headings where the green is chosen since it is the closest heading to the regressed vector.

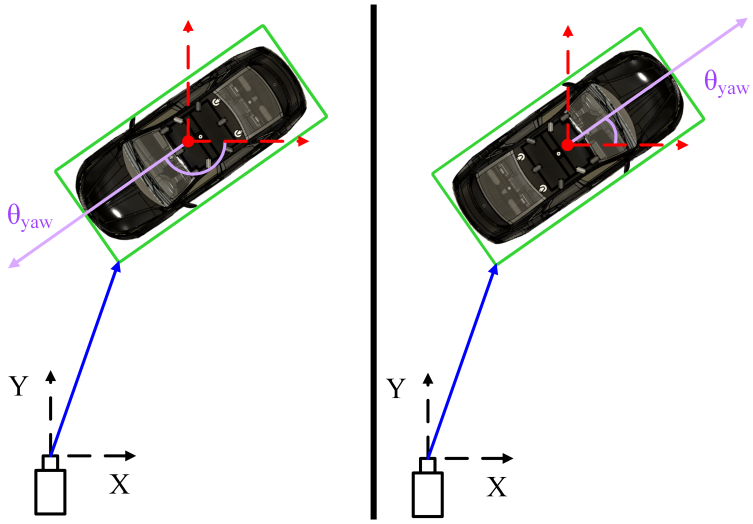


Figure 3.8: A visual representation of the 3D detection problem from BEV. **Green:** The bounding box used to determine the IoU overlap in the computation of the *average precision*. **Blue:** The vector used to compute *the average error in distance to impact*. **Red:** The centroid of the bounding box used to calculate the *average error in estimating the centroid*. The importance of explicit orientation estimation can be seen as an object's bounding box does not change when the orientation (**purple**) is shifted by $\pm\pi$ radians.

Generating Final Detections

Similar to the RPN, the inputs to the multiview detection network are feature crops generated from projecting the proposals into the two input views. As the number of proposals is an order of magnitude lower than the number of anchors, the original feature map with a depth of 256 is used for generating these feature crops. Crops from both input views are resized to 7×7 and then fused with an element-wise mean operation. A *single* set of three fully connected layers of size 2048 process the fused feature crops to output box regression, orientation estimation, and category classification for each proposal. Similar to the RPN, we employ a multi-task loss combining two Smooth L1 losses for the bounding box and orientation vector regression tasks, and a cross-entropy loss for the classification task as shown in Eq. 3.3. Final detected boxes are only considered in the evaluation of the regression loss if they have at least a 0.65 or 0.55 2D IoU in BEV with the ground truth boxes for the *car* and *pedestrian/cyclist* classes, respectively. Since multiple proposals can be regressed to the same space in BEV, we perform 2D NMS with an IoU threshold of 0.01 to eliminate overlapping detections. The final loss is calculated by the sum of the RPN loss described in Eq.3.2 and Eq.3.3.

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) + \lambda \frac{1}{N_{ang}} \sum_i p_i^* L_{ang}(t_i, t_i^*) \tag{3.3}$$

3.1.6 Training

We train two networks, one for the *car* class and one for both the *pedestrian* and *cyclist* classes. The RPN and the detection networks are trained jointly in an end-to-end fashion using mini-batches containing one image with 512 and 1024 ROIs, respectively. The network is trained for 120K iterations using an ADAM optimizer [39] with an initial learning rate of 0.0001 that is decayed exponentially every 100K iterations with a decay factor of 0.1. For regularization, we employ dropout with a probability of 0.5 throughout all the fully connected layers. Batch Normalization is also applied to all *FC* layers. Batch normalization leads to improvements in convergence while eliminating the need for other forms of regularization.

Path-Drop

Path-drop training is applied by implementing the following. At each iteration, we randomly generate 3 probabilities, the first determines the chance of keeping the image branch, the second determines keeping the BEV branch, and the third makes the final decision in the case where both branches are killed off. So if our random probability is higher than the probability set as hyperparameter, that branch is dropped. For instance at each iteration we decide to drop either the BEV or image branch but we never drop both. Although dropping one input during training might be an ideal technique to make the network robust to missing inputs such as lack of image or BEV, overall our experiments showed that the results were insensitive to path-drop. Therefore in all our experiments, the probability of dropping either branch was set to 0.9 indicating that both branches were kept during training most of the time.

Data Augmentation

Data augmentation techniques such as flipping and PCA jittering are applied to the training data. Data augmentation helps to increase the training instances, and in turn, it reduces overfitting. Both image and point cloud data can be flipped horizontally where the corresponding labels are also flipped to match the flipped object location. PCA jittering consists of altering the intensities of the RGB channels in the training images. This is implemented by computing PCA on all RGB values in the training data, and then adding multiples of the found principle components to the images, as described in [40].

3.2 Proposed Evaluation Metric

This section discusses the proposed metric to measure the performance of heading estimation.

3.2.1 The Average Heading Similarity Metric

In addition to the metrics discussed in Section 2.5, AVOD 3D detection results are also evaluated using the 3D and BEV AP and Average Heading Similarity (AHS) at 0.7 IoU threshold for the car class, and 0.5 IoU threshold for the pedestrian and cyclist classes. The AHS is the Average Orientation Similarity (AOS) [23], but evaluated using 3D IOU and

global orientation angle instead of $2D$ IOU and observation angle, removing the metric's dependence on localization accuracy.

$$AHS = \frac{1}{11} \sum_{r \in (0, 0.1, \ldots 1)} \max_{\hat{r}: \hat{r} \geq r} s(\hat{r}), \tag{3.4}$$

where $s(r)$ is defined at every recall value r according to:

$$s(r) = \frac{1}{|\mathbb{D}(r)|} \sum_{i \in \mathbb{D}(r)} 1(IoU \geq \lambda) \frac{1 + \cos(\theta - \theta^*)}{2}. \tag{3.5}$$

Here, θ is the global orientation estimate, θ^* is the ground truth global orientation, and $\mathbb{D}(r)$ is the set of all detections at recall r. $1(IOU \geq \lambda)$ is an indicator function used to only consider valid detections in the computation of the heading similarity. The AHS can be evaluated in either 3D or BEV spaces by switching the source of the IoU computation in the indicator function.

3.3 Experiments and Results

We test AVOD's performance on the proposal generation and object detection tasks on the three classes of the KITTI Object Detection Benchmark [23]. We follow [6] to split the provided 7481 training frames into a *training* and a *validation* set at approximately a $1:1$ ratio. For submission to the KITTI server, we train on our custom split with 85% of training data used for training and the rest for validation. Note that since there are significantly fewer instances of pedestrian and cyclists compared to the car instances, this split in particular, helps to achieve better generalization on the detection of both pedestrian and cyclist classes by including more instances in the training set. For evaluation, we follow the *easy, medium, hard* difficulty classification proposed by KITTI.

3D Proposal Recall: 3D proposal generation is evaluated using *3D bounding box recall* at a 0.5 3D IoU threshold. We compare our RPN against the proposal generation algorithms 3DOP [4] and MONO3D [3]. Figure 3.9 shows the recall vs number of proposals curves for our fusion RPN, 3DOP and Mono3D. It can be seen that our fusion based RPN outperforms both 3DOP and Mono3D by a wide margin on all three classes. As an example, our RPN achieves an 86% 3D recall on the Car class with just 10 proposals per frame. The maximum recall achieved by 3DOP and Mono3D on the Car class is 73.87% and 65.74% respectively.

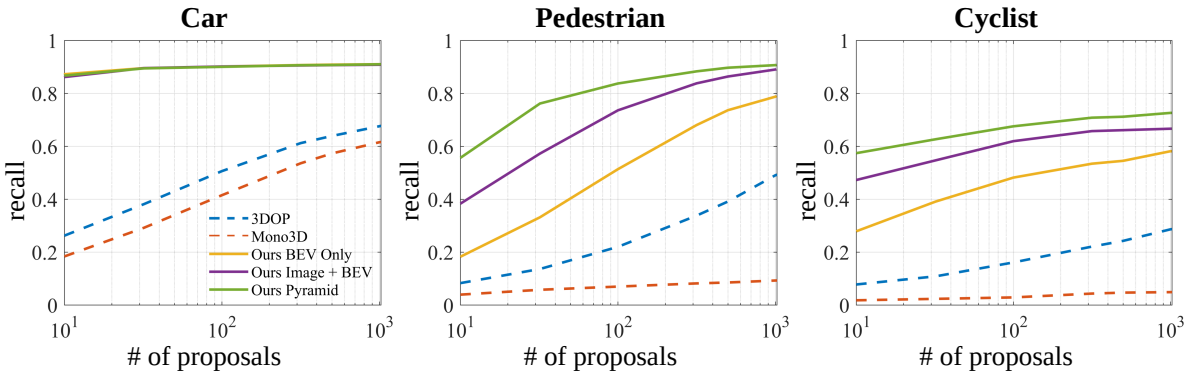


Figure 3.9: The recall vs number of proposals at a 3D IoU threshold of 0.5 for the three classes evaluated on the *validation* set at *moderate* difficulty.

This gap is also present for the pedestrian and cyclist classes, where our RPN achieves more than 20% increase in recall at 1024 proposals. This large gap in performance suggests the superiority of learning based approaches over methods based on hand crafted features. For the car class, our RPN achieves a 91% recall at just 50 proposals, whereas MV3D [6] reported requiring 300 proposals to achieve the same recall. It should be noted that MV3D does not publicly provide proposal results for cars, and was not tested on pedestrians or cyclists.

3D Object Detection: 3D detection results are evaluated using the 3D AP, BEV AP, and AHS metrics at 0.7 IoU threshold for the car class, and 0.5 IoU threshold for the pedestrian and cyclist classes. We compare against publicly provided results from MV3D [6] and Deep3DBox [50] on the *validation* set. It has to be noted that no currently published method provides public results on the pedestrian and cyclist classes for the 3D object detection task, and hence comparison is done for the *car* class only. On the *validation* set (Table 3.5), AVOD is shown to outperform MV3D by 1.84% AP on the *moderate* setting and 3.22% on the *hard* setting. However, AVOD achieves a 30.28% and 27.54% increase in AHS over MV3D at the *moderate* and *hard* setting respectively. This can be attributed to the loss of orientation vector direction discussed in Section 3.1.5 resulting in orientation estimation up to an additive error of $\pm\pi$ radians. To verify this assertion, Figure 3.11 shows a qualitative comparison of the results of AVOD and MV3D in comparison to KITTI's ground truth. It can be seen that MV3D assigns erroneous orientations for almost half of the cars shown. On the other hand, AVOD assigns the correct orientation for all cars in that particular scene. As expected, the gap in 3D localization performance between

Deep3DBox and AVOD is very large. It can be seen in Figure 3.11 that Deep3DBox fails at accurately localizing most of the vehicles in 3D. This further enforces the superiority of fusion based methods over monocular based ones. We also compare the performance of our architecture on the KITTI *test* set with MV3D, VoxelNet[76], and F-PointNet[54]. *Test* set results are provided directly by the evaluation server, which does not compute the AHS metric. Table 3.6 shows the results of AVOD on KITTI's *test* set. It can be seen that with only the modified VGG feature extractor, AVOD performs quite well on all three classes, while being *twice* as fast as the next fastest method, F-PointNet. However, once we add our high-resolution feature extractor, i.e. feature pyramid architecture, AVOD-FPN outperforms all other methods on the car class in 3D object detection, with a noticeable margin of 4.19% on hard (highly occluded or far) instances in comparison to the second best performing method, F-PointNet. Finally, our network that is trained on only the KITTI training data outperforms the ImageNet pretrained F-PointNet on the car class and achieves comparable results on *pedestrians* and *cyclist* classes while being 1.7× faster. MV3D, F-PointNet, and VoxelNet do not provide orientation estimation results to be evaluated using KITTI's 2D AOS metric. Thus, performance comparison in terms of heading estimation with existing methods cannot be performed.

	Easy		Moderate		Hard	
	AP	AHS	AP	AHS	AP	AHS
Deep3DBox	5.84	5.84	4.09	4.09	3.83	3.83
MV3D	83.87	52.74	72.35	43.75	64.56	39.86
AVOD-FPN	**84.41**	**84.19**	**74.44**	**74.11**	**68.65**	**68.28**

Table 3.5: A comparison of the performance of Deep3DBox [50], MV3D [6], and AVOD-FPN evaluated on the *car* class in the *validation* set. For evaluation, we show the AP and AHS (in %) at 0.7 3D IoU.

Runtime and Memory Requirements: We use FLOP count and number of parameters to assess the computational efficiency and the memory requirements. AVOD-FPN architecture employs roughly 38.073 million parameters, approximately 16% that of MV3D. The *deep fusion* scheme employed by MV3D triples the number of fully connected layers required for the second stage detection network, which explains the significant reduction in the number of parameters of AVOD-FPN. Furthermore, AVOD-FPN architecture requires 231.263 billion FLOPs per frame allowing it to process frames in 0.1 seconds on a TITAN Xp GPU, taking $20ms$ for pre-processing, where the pre-processing includes operations such

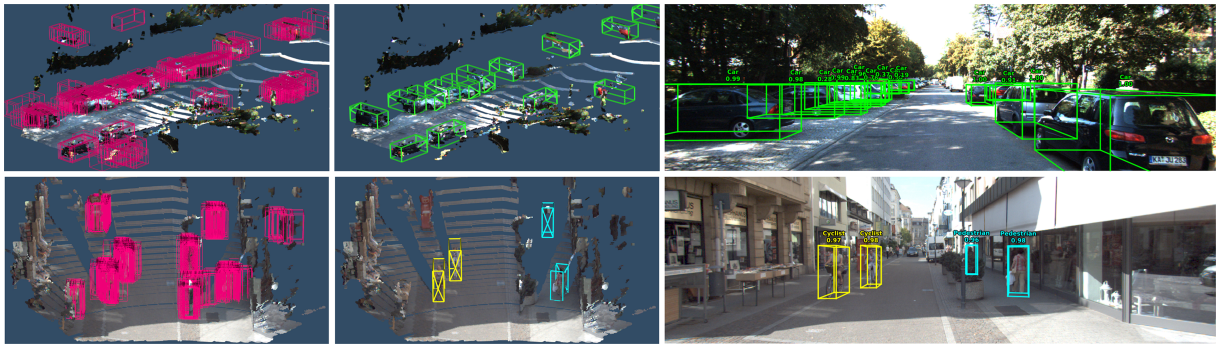


Figure 3.10: Qualitative results of AVOD for cars (top) and pedestrians/cyclists (bottom). **Left:** 3D proposal network output, **Middle:** 3D detection output, and **Right:** the projection of the detection output into image. The 3D LIDAR point cloud has been colorized and interpolated for better visualization.

as loading the input, BEV map generation, anchor generation and empty anchor filtering, and $80ms$ for inference. This makes it $1.7\times$ faster than F-PointNet, while maintaining state-of-the-art results. Finally, both AVOD and AVOD-FPN require only 2 gigabytes of GPU memory at inference time, making them suitable to be used for deployment on autonomous vehicles.

3.3.1 Ablation Studies

Table 3.7 shows the effect of varying different hyperparameters on the performance measured by the AP and AHS, number of model parameters, and FLOP count of the proposed architecture. The original network uses hyperparameter values described throughout the thesis up to this point, along with the feature extractor of MV3D. We study the effect of the RPN's input feature vector origin and size on both the proposal recall and final detection AP by training two networks, one using *BEV only* features, and the other using feature crops of size 1×1 as input to the RPN stage. We also study the effect of different bounding box encoding schemes shown in Figure 3.5, and the effects of adding an orientation regression output layer on the final detection performance in terms of AP and AHS. Finally, we compare different fusion schemes proposed previously by MV3D [6]. The fusion schemes are shown in Figure 3.12.

RPN Input Variations: Figure 3.9 shows the recall vs. number of proposals curves

			AP_{3D} (%)			AP_{BEV}(%)		
Method	Runtime (s)	Class	Easy	Moderate	Hard	Easy	Moderate	Hard
MV3D [6]	0.36	**Car**	71.09	62.35	55.12	86.02	76.90	68.49
VoxelNet [76]	0.23		77.47	65.11	57.73	**89.35**	79.26	77.39
F-PointNet [54]	0.17		81.20	70.39	62.19	88.70	84.00	75.33
AVOD	**0.08**		73.59	65.78	58.38	86.80	**85.44**	77.73
AVOD-FPN	0.1		**81.94**	**71.88**	**66.38**	88.53	83.79	**77.90**
VoxelNet [76]	0.23	**Ped.**	39.48	33.69	31.51	46.13	40.74	38.11
F-PointNet [54]	0.17		**51.21**	**44.89**	40.23	58.09	50.22	47.20
AVOD	**0.08**		38.28	31.51	26.98	42.51	35.24	33.97
AVOD-FPN	0.1		50.80	42.81	**40.88**	**58.75**	**51.05**	**47.54**
VoxelNet [76]	0.23	**Cyc.**	61.22	48.36	44.37	66.70	54.76	50.55
F-PointNet [54]	0.17		**71.96**	**56.77**	**50.39**	**75.38**	**61.96**	**54.68**
AVOD	**0.08**		60.11	44.90	38.80	63.66	47.74	46.55
AVOD-FPN	0.1		64.00	52.18	46.61	68.09	57.48	50.77

Table 3.6: A comparison of the performance of AVOD with the state-of-the-art 3D object detectors evaluated on KITTI's *test* set. Results are generated by KITTI's evaluation server [23].

for both the original RPN and BEV only RPN on the three classes on the *validation* set. For the pedestrian and cyclist classes, fusing features from both views at the RPN stage is shown to provide a 10.1% and 8.6% increase in recall over the BEV only version at 1024 proposals. This difference increases as the number of proposals is decreased reaching 20% when using 100 proposals. For the car class, adding image features as an input to the RPN does not seem to provide a higher recall value over the BEV only version. We attribute this to the fact that instances from the car class usually occupy a large space in the input BEV map, providing sufficient features in the corresponding output feature map to reliably generate object proposals. The effect of the increase in proposal recall on the final detection performance can be observed in Table 3.7.

Using both image and BEV features at the RPN stage results in a 6.9% and 9.4% increase in AP over the BEV only version for the pedestrian and cyclist classes respectively. We perform an additional experiment using 1 pixel feature vector size as an input to the RPN from both views. We observe a drop in AP for all three classes, with the drop being much higher for the pedestrian and cyclist classes (26.9% and 32.6% decrease for pedestrians and cyclists vs. a 9.5% decrease for cars). This enforces the hypothesis that a single pixel does not contain sufficient information for proposal generation for these smaller classes. It has to be noted that the BEV only RPN requires 266.64 million less

Architecture	Car		Pedestrian		Cyclist		Number Of Parameters	FLOPs
	AP_{3D}	AHS_{3D}	AP_{3D}	AHS_{3D}	AP_{3D}	AHS_{3D}		
Original	74.1	73.9	39.5	29.8	41.6	33.2	38,073,528	186,284,945,569
RPN BEV Only	74.1	73.9	32.6	26.7	32.2	30.2	0	**-266,641,569**
Feature Pyramid Extractor	**74.8**	**74.5**	**58.8**	**43.3**	**49.7**	**48.7**	**-21,391,104**	+44,978,386,776
RPN 1 Pixel Feature Size	64.6	64.3	12.6	11.6	9.0	8.2	-4,096	-20,480
Axis Aligned	67.6	67.5	32.6	27.8	36.6	35.6	-8196	-40,958
4 Corners No Orientation	67.8	43.1	37.8	17.9	41.1	18.6	-4,098	-20,418
8 Corners No Orientation	66.9	34.1	37.8	18.1	40.4	21.0	+24638	+122,879
8 Corners Ordered No Orientation	51.6	51.6	18.3	16.1	13.2	13.1	+24638	+122,879
Late Fusion	73.2	72.6	38.0	29.9	36.8	34.8	+34,113,536	+170,536,962
Deep Fusion	72.6	72.4	36.4	27.6	36.9	32.2	+34,113,536	+170,536,962
Basic FC Layers(2048)	67.9	67.5	36.2	26.4	32.4	27.0	+68,169,728	+266,641,569

Table 3.7: A comparison of the performance of different variations of hyperparameters, evaluated on the *validation* set at *moderate* difficulty. We use a 3D IoU threshold of 0.7 for the car class, and 0.5 for the pedestrian and cyclist classes. The effect of variation of hyperparameters on the FLOPs and number of parameters are measured relative to the original network.

FLOPs than the original network as it does not need to generate feature crops from image view feature maps. The decrease in the FLOP count for the network does not compensate for the loss in performance.

Bounding Box Encoding: We study the effect of different bounding box encodings shown in Figure 3.5 by training two additional networks. The first network estimates axis aligned bounding boxes, using the regressed orientation vector as the final box orientation. The second and the third networks use our 4 corner and MV3D's 8 corner encodings without additional orientation estimation as described in Section 3.1.5. As expected, without orientation regression to provide orientation angle correction, the two networks employing the 4 corner and the 8 corner encodings provide a much lower AHS than the original network for all three classes. This phenomenon can be attributed to the loss of orientation information as described in Section 3.1.5.

Fusion Methods: Figure 3.12 shows the different fusion methods implemented for the second stage, and compared in this work. The choice of using early fusion for our architecture was due to no observed improvement when using late or deep fusion proposed by [6]. On the other hand, early fusion requires half the number of parameters in the second stage fully connected layers, and as such requires much less memory when deployed. Early fusion is also marginally faster at inference time. The performance of fusion methods is shown in

Table 3.7. Note that *Basic FC Layers* means no fusion was applied and 3 separate task specific branches were used.

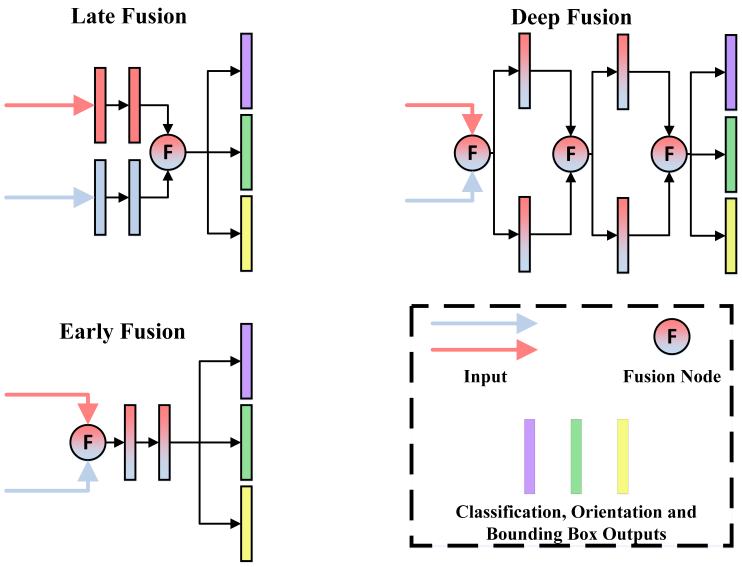


Figure 3.12: The different fusion schemes used for comparison. The element-wise *mean* operation is used for all fusion modes.

Qualitative Results: Figure 3.10 shows the output of the RPN and the final detections in both 3D and image space. To show generalization, we also deploy AVOD on our autonomous vehicle with a different set of cameras (Ximea MQ013CG-E2 1.3 MP) and a lower resolution LIDAR (Velodyne HDL-32e) than that used in KITTI. Results of generalization are shown in Figure 3.13.

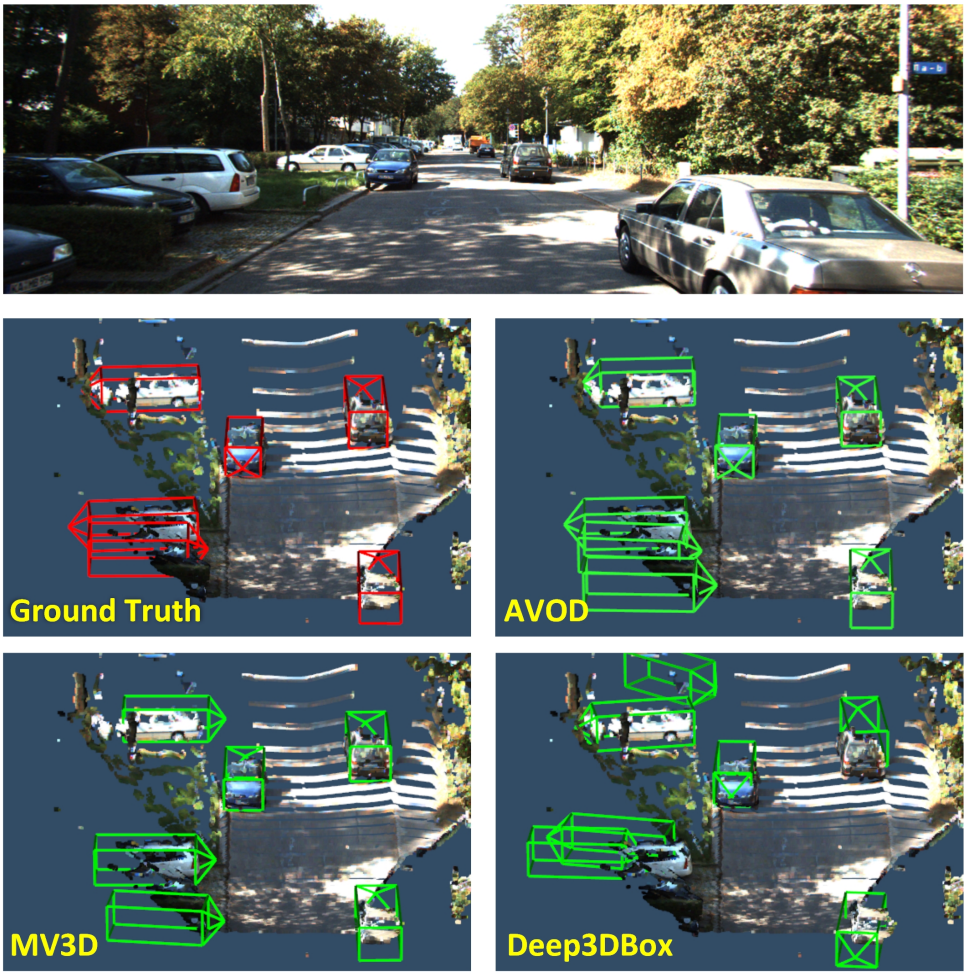


Figure 3.11: A qualitative comparison between MV3D [6], Deep3DBox [50], and AVOD relative to KITTI's ground truth on a sample in the *validation* set. It can be noted that MV3D can only determine the global orientation of objects with an additive error of $\pm\pi$ radians.

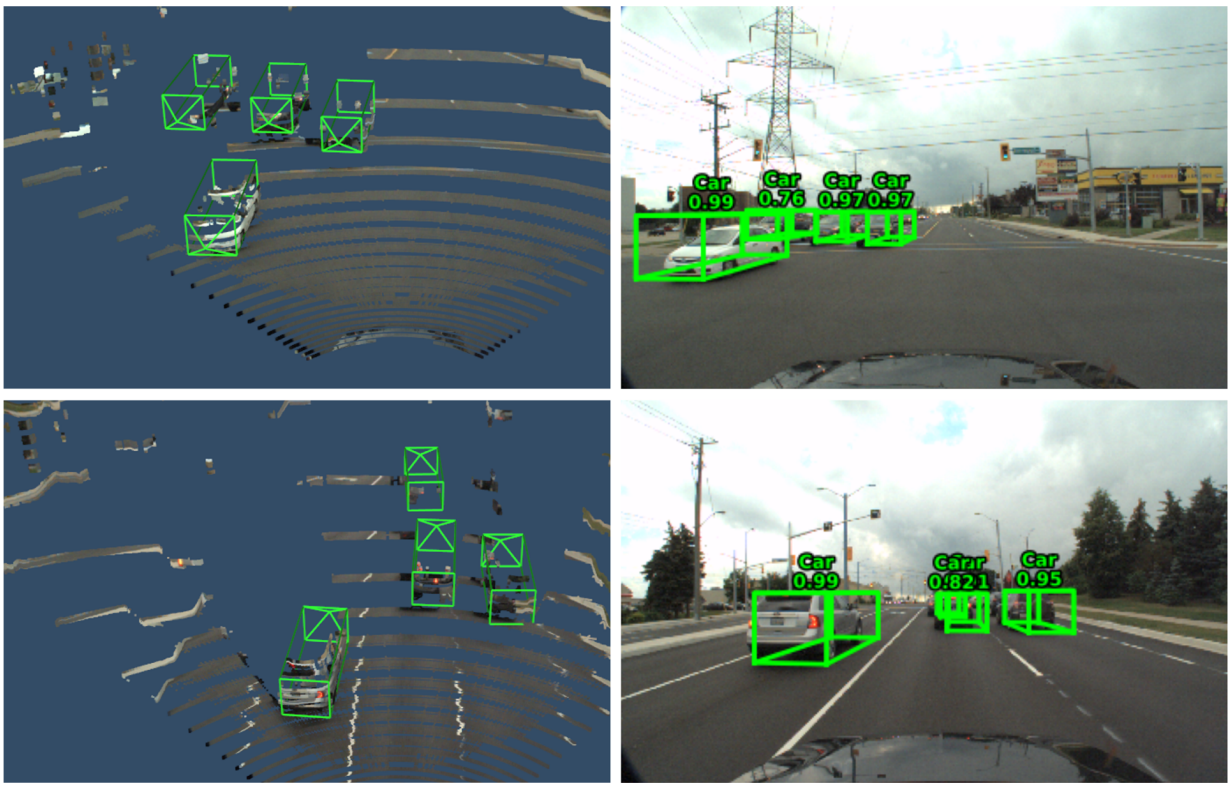


Figure 3.13: Results of deploying AVOD on our autonomous vehicle. It can be seen that the network generalizes well to new scenes even when using data from a different camera and a lower resolution LIDAR sensor.

Chapter 4

Single Stage Detection

4.1 Motivation

This chapter discusses AVOD-SSD, a 3D single stage detector for object detection. The highest accuracy object detectors to date are based on two-stage detectors. In contrast, single stage detectors have the potential to be faster and simpler architecture-wise. Single stage object detection algorithms are simple in a sense that they completely eliminate proposal generation and subsequent feature resampling stage, and thus encapsulate all computation into one stage. With this in mind, this work further explores making the necessary modifications to AVOD to operate as a single stage detector, while still maintaining a fairly high accuracy. Furthermore the speed accuracy trade-offs between AVOD, AVOD-FPN and AVOD-SSD are discussed.

4.2 The AVOD-SSD Architecture

A single stage detector simply skips the proposal generation part, but rather regresses the initial anchors directly. The architecture is depicted in Figure 4.1. The architecture is similar to AVOD with the following modifications. At the RPN stage, the 1×1 Conv is removed since the network requires a higher resolution feature map in order to regress the anchors directly. Similar to AVOD's second stage network, crops from both input views are resized to 7×7 and then fused with an element-wise mean operation. After the fusion of cropped features, the same task-specific *FC* layers previously used in AVOD to perform classification, regression and orientation, are deployed to classify and regress the anchors

directly. The structure of the *FC* layers is shown in Figure 4.2. The second stage bounding box refinement is removed and NMS is then applied on all the regressed anchors resulting to the final bounding boxes. When making these modifications, we need to consider two factors. First, since we are now regressing all the anchors, we are facing a class imbalance scenario. This is due to having the majority of the anchors as background and only very few anchors containing actual objects. This can potentially stop the single stage detector from achieving state-of-the-art accuracy. The authors in [47] proposed a new loss function to eliminate this barrier, and the effect of this loss function is demonstrated in Section 4.3. The second factor affecting the performance is the amount of computation happening inside the network from an implementation point of view. Note that with AVOD, non-oriented proposals are being regressed at the RPN stage. These regressed anchors then need to be converted to the desired final box format (such as *4 Corners + Heights* or *8-Corners*) to perform the regression at the second stage. However with AVOD-SSD this box conversion processing can be pushed onto the pre-processing stage. Even with these code optimizations, the computation cost of extracting and fusing features, along with the regression of large number of anchors for classification, regression and orientation is still too high to achieve a significant boost in speed. This is further discussed in Section 4.3.

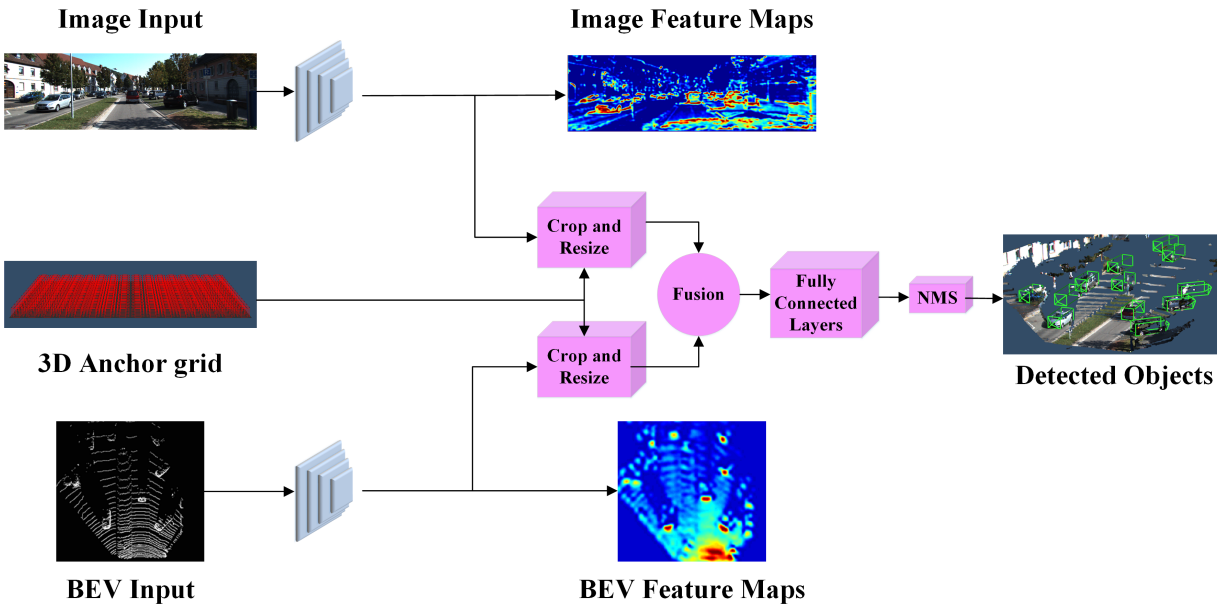


Figure 4.1: AVOD-SSD's architectural diagram. The feature extractors are shown in **blue** and the single stage detection network in **pink**.

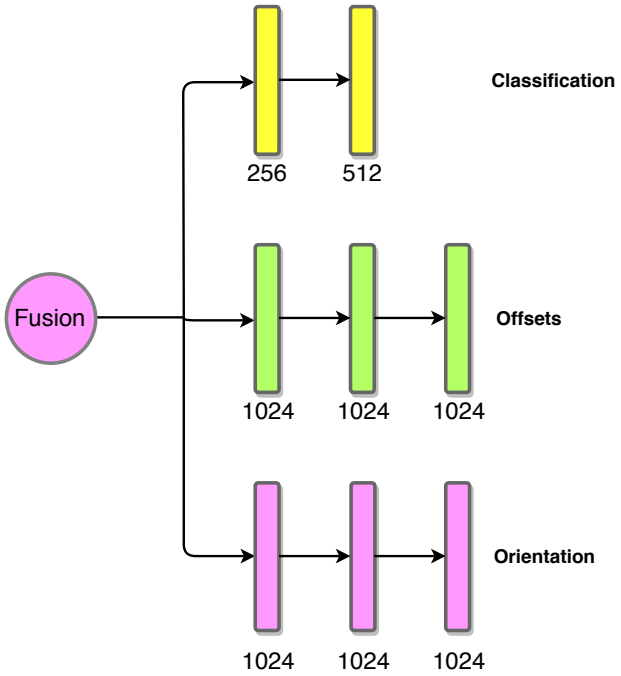


Figure 4.2: AVOD-SSD *FC* layers. The number at the bottom of each layer indicates the size of the layer.

4.2.1 Focal Loss

Single stage object detection methods typically face a large class imbalance during training. This is because these detectors evaluate $10K - 100K$ candidate locations (i.e. anchors) per frame but only a few locations contain objects. This imbalance causes two problems: (1) training is extremely inefficient as most locations are non-objects and the network will spent most of its time training on negative samples; (2) The easy negatives can simply overwhelm the network. One common solution proposed is to perform hard negative mining [68, 64] that samples hard examples during training or adopt a more complex sampling technique. Authors in [47] proposed a focal loss function that handles the class imbalance faced by a single stage detector. This enables the network to efficiently train on all examples without requiring any complex sampling. In addition the proposed loss function reduces the effect of easy negatives on the computation of the gradients.

Focal loss is designed to address the class imbalance issue by down-weighting easy examples. This way their contribution to the total loss is small even if their number is

large. Focal loss essentially performs the *opposite* role of a robust loss function (e.g. Huber loss [31]), which reduces the contribution of outliers by down-weighting the loss of examples with large errors, i.e. hard examples, and it's commonly used in neural networks. Focal loss instead focuses the training on a sparse set of hard examples. In particular it addresses the scenario where there is an extreme imbalance between foreground and background classes during training (e.g. $1 : 1000$). This happens in a single stage detector where out of thousands of anchors, there might be only a few locations containing objects.

In order to introduce the formulation of focal loss, we start with cross entropy (CE) loss for binary classification:

$$CE(p, y) = \begin{cases} -log(p), & \text{if } y = 1 \\ -log(1-p), & \text{otherwise} \end{cases} \tag{4.1}$$

In the above equation $y \in \{\pm 1\}$ specifies the ground-truth class and $p \in [0, 1]$ is the model's estimated probability for the class with label $y = 1$. For notational convenience, we define p_t:

$$p_t = \begin{cases} p, & \text{if } y = 1 \\ 1-p, & \text{otherwise} \end{cases} \tag{4.2}$$

and then we can rewrite $\text{CE}(p, y) = \text{CE}(p_t) = -log(p_t)$.

Balanced Cross Entropy

A common technique to address the class imbalance issue via the CE loss function is to introduce a weighting factor $\alpha \in [0, 1]$ for class 1 and $1 - \alpha$ for class -1, where α is a hyperparameter that needs to be tuned for the network. For notational convenience similarly applied to p_t, we define α_t. We write the α-Balanced CE loss as :

$$BCE(p_t) = -\alpha_t log(p_t) \tag{4.3}$$

Focal loss then builds on this simple extension.

Focal Loss Definition

As previously discussed, the large class imbalance during training overwhelms the cross entropy loss. Due to such imbalance, the majority of the loss and the gradients are dominated by the easily classified negative examples. Although α balances the importance of

positive vs. negative examples, it does not differentiate between easy vs. hard examples. Instead focal loss down-weights easy examples and focuses the training on hard negatives.

Focal loss uses a modulating factor $(1-p_t)^\gamma$ with the cross entropy loss, with a tunable *focusing* parameter $\gamma \geq 0$.

$$FL(p_t) = -(1-p_t)^\gamma log(p_t) \tag{4.4}$$

The focal loss is visualized for several values of $\gamma \in [0, 5]$ in Figure 4.3. Two important properties of this loss are the following: (1) When an example is misclassified and p_t is small, the modulating factor is near 1 and hence the loss is not affected. As $p_t \to 1$, the factor goes to 0 and the loss for correctly classified examples is down-weighted. (2) The focusing parameter γ adjusts the rate at which easy examples are down-weighted. When $\gamma = 0$, FL is equivalent to CE, and as γ is increased the effect of the modulating factor increases. We can think of the modulating factor γ as reducing the loss contribution from easy examples while extending the range in which an example receives low loss. For instance, with $\gamma = 2$, an example classified with $p_t = 0.9$ would have $\approx 100\times$ lower loss compared with CE and with $p_t = 0.968$ it would have $\approx 1000\times$ lower loss. This increases the importance of correcting misclassified examples by down-scaling the loss by at most $4\times$ for $p_t \leq .5$ and $\gamma = 2$ [47]. An α-balanced variant of the focal loss can also be defined as:

$$FL(p_t) = -\alpha_t(1-p_t)^\gamma log(p_t) \tag{4.5}$$

This provides a slight improvement in accuracy over the non-α-balanced form.

Figure 4.3 shows the properties of both loss functions, however for CE loss, it can be seen that even examples that are easily classified ($p_t \gg .5$) incur a loss with a large magnitude. When the loss is summed over a large number of easy examples, these small values can overwhelm the rare class.

Implementation of focal loss combines the sigmoid operation for computing p. Extending the focal loss to the multi-class domain is also possible; however, for, simplicity this work focuses on the binary loss, and this loss is applied for the case of Car detection where there are only two classes: Car and Background classes.

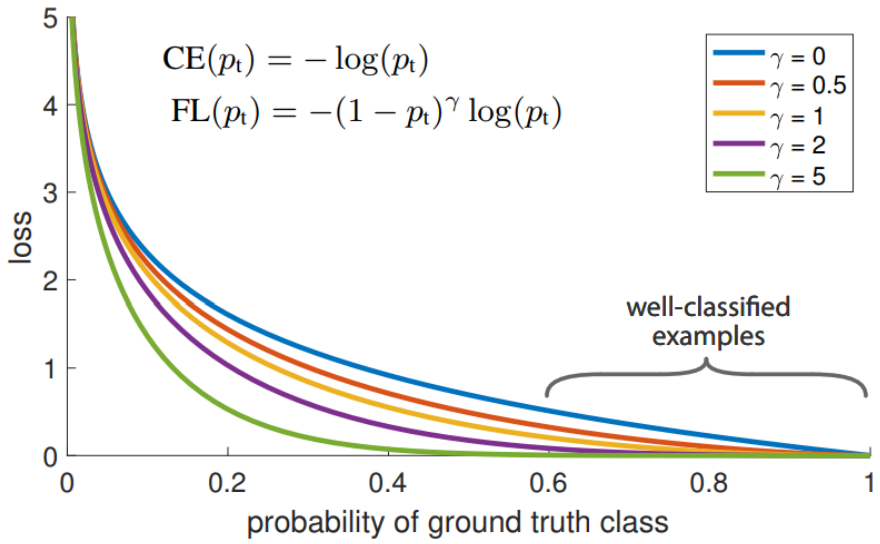


Figure 4.3: The focal loss is visualized for several values of $\gamma \in [0, 5]$ [47].

4.3 Experiments

4.3.1 Performance & Speed Analysis

Table 4.1 shows the performance comparison of AVOD, AVOD-FPN and AVOD-SSD. Table 4.2 demonstrates the effect of the focal loss applied to AVOD-SSD, increasing the AP from 61.52 to 73.30. Note that the AP is evaluated at both 0.7 IoU and 0.5 IoU threshold, where a 0.7 IoU threshold is more descriptive in terms of localization accuracy. For instance $AP_{3D}(0.7)$ indicates the AP was evaluated at a 0.7 IoU threshold. In this table the "Original" refers to AVOD-SSD with feature pyramid feature extractor, Basic FC layers and focal loss.

Both AVOD and AVOD-FPN are optimized for accuracy. For instance, they use two clusters for car detection, which is tailored towards detecting smaller cars such as smart cars. Table 4.2 show various hyperparameters selected for AVOD-SSD that affect the performance and some can potentially reduce run-time speed [1]. Note that some of the

[1]Note that the run-time speeds are approximations. The run-time speed is typically averaged over 100 iterations and the speed can vary when run on different machines. All the speed profiling was done on a single machine for a fair comparison.

network parameter optimizations are applicable to all three architectures as they all share the bulk of the computation overhead. Table 4.1 also shows an additional experiment, where instead of reducing the computational load on the network, we increase the BEV resolution to study the effect on the performance (referred to as *High Res BEV*). For this experiment, the point cloud is discretized with a '0.07' meters resolution resulting to a BEV map of size 1000×1144. This shows that increasing the resolution of BEV improves the AP of AVOD-SSD while increasing the runtime, while on AVOD-FPN the same improvement is not achieved. This might indicate that AVOD-SSD can benefit from a richer feature representation.

Overall it can be seen that by making modifications such as adopting focal loss and changing other hyperparameters such as using Basic *FC* layers, i.e. separate task specific layers as opposed to Fusion layers, and fine-tuning the loss function parameters, AVOD-SSD can achieve similar accuracy to AVOD. Note that Early Fusion was adopted for both AVOD and AVOD-FPN because of better performance; however it did not perform as well with the AVOD-SSD architecture. Instead, separate task specific layers with reduced layer sizes seems to provide the best performance. However during these experiments, it was found that reducing the size of the Basic *FC* layers in general does help with the performance on AVOD as shown in Table 4.1, compared with the results presented in Table 3.7 where larger sized layers (2048) were used.

In terms of the choice of the feature extractor, that is between the modified VGG and feature pyramids, AVOD-SSD with VGG feature extractor performs poorly with an increased inference speed. The poor performance might be due to the fact that the network requires a rich and multi-scale feature representation such as feature pyramids to be able to regress anchors directly. The speed is related to the fact that the 1×1 Conv dimensionality reduction is removed, and hence the network needs to process the crops along the entire dimension of 256 which results in high memory consumption as these crops need to be stored in memory as well as increase in the processing run-time. However with feature pyramid feature extractor, the dimension of features along depth is only 32 and hence, it runs faster.

Although single stage detectors promise a speed boost as they skip the proposal generation step, saving significant computation time in the case of 2D detection, the same speed boost was not achieved in the case of 3D detection. AVOD-SSD is only $0.01s$ faster than AVOD-FPN while being $0.01s$ slower than AVOD. This is due to various overhead in terms of certain computations. One large computational overhead in all variations of AVOD architecture is the feature extractor stage where the convolution processing is extensive, specially in the case of the feature pyramid layers. Interestingly, the computational overhead of processing large number of anchors is smaller than the number and size of

the feature extractor layers. This was confirmed by performing a small experiment where the anchor stride is increased significantly resulting in a small number of anchors. Even significantly reducing the number of anchors does not lead to a significant speed boost in the GPU processing time.

Finally Figures 4.4 and 4.5 show the Precision-Recall curve for AVOD-FPN vs. AVOD-SSD for 3D detection, BEV detection and 2D detection on the *validation* and *test* set, respectively. Note that we compare performance against AVOD-FPN since it has better performance compared to the original AVOD architecture. It can be seen that in terms of recall, both networks are performing well, however, AVOD-FPN achieves a slightly higher precision due to the second stage refinement process. The interpretation of these plots is that AVOD-FPN in general is more robust towards FPs, whereas AVOD-SSD is more prone to FPs. In addition AVOD-FPN has more precise localization accuracy. This is largely due to the fact that with two-stage detectors, the network has two chances at both classification and regression. First the RPN distinguishes between an object and non-object, and the second stage further decides on the class of the potential objects as well as refining the location of the bounding box. In contrast with single stage detector, the network only has one pass at deciding on the class of a potential object and hence the chances of detecting FPs increases. Overall the AP of AVOD and AVOD-SSD are fairly similar when evaluated on the *validation* set, indicating that a single stage detector can perform as well as a two-stage detector with two-stage detector being more robust towards FPs and slightly more accurate in terms of localization. Table 4.4 shows the performance of AVOD-SSD on the KITTI's evaluation server compared against AVOD and AVOD-FPN. Finally Table 4.3 shows the effect of varying focal loss parameters γ and α. Similar results in terms of the best values for γ and α providing the highest accuracy reported in [47] was found in these experiments.

	Car						
Architecture	$AP_{3D}(0.7)$	$AHS_{3D}(0.7)$	$AP_{3D}(0.5)$	$AHS_{3D}(0.5)$	Number Of Parameters	FLOPs	Runtime speed (s)
AVOD	74.23	73.97	89.10	**88.75**	38,073,528	186,284,945,569	**0.08**
AVOD-FPN	**74.81**	**74.53**	89.11	88.63	-21,391,104	+27,422,011,231	0.1
AVOD-FPN (Basic FC Layers - 1024)	73.08	72.54	**89.51**	88.67	-24,534,784	+27,422,011,231	0.09
AVOD-FPN(High Res BEV)	73.74	73.11	89.07	88.02	-21,391,104	+152,577,646,431	0.14
AVOD-SSD(VGG)	56.11	55.89	83.21	82.67	-139,274	**-46,616,465,569**	0.2
AVOD-SSD(FPN)	73.93	73.32	88.94	87.96	**-25,206,346**	+27,358,523,231	0.09
AVOD-SSD(FPN + High Res BEV)	74.53	74.08	88.91	88.19	-24,673,610	+152,476,782,431	0.16

Table 4.1: Performance analysis of AVOD, AVOD-FPN and AVOD-SSD on the *validation* set and *moderate* difficulty.

Architecture	Car AP_{3D}	Car AHS_{3D}	Car $AP_{3D}(0.5)$	Car $AHS_{3D}(0.5)$	Number Of Parameters	FLOPs	Runtime speed (s)
Original	**73.93**	**73.32**	**88.94**	**87.96**	12,867,182	213,643,468,800	0.09
Basic FC Layers without Focal Loss	61.52	60.32	87.78	85.70	0	0	0.09
Early Fusion without Focal Loss	58.88	58.50	84.87	84.03	+3,676,416	0	0.1
Early Fusion (2048)	67.02	66.81	84.64	84.24	+3,676,416	0	0.1
1 Cluster	67.60	67.02	88.14	87.17	0	0	0.08
Reduced Image Resolution	63.86	63.63	82.39	81.93	-3,676,416	-75,220,992,000	0.08

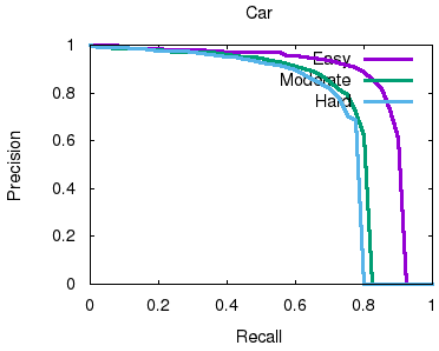
Table 4.2: Effect of different parameters on the accuracy and speed of the AVOD-SSD on the *validation* set and *moderate* difficulty.

γ	α	Car $AP_{3D}(0.7)$	Car $AHS_{3D}(0.7)$	Car $AP_{3D}(0.5)$	Car $AHS_{3D}(0.5)$
1.0	.25	70.85	70.41	88.76	87.86
2.0	.25	**73.93**	**73.32**	**88.94**	**87.96**
2.0	.75	72.10	71.34	88.46	87.70
2.0	.5	68.01	67.70	88.53	87.86
3.0	.25	70.03	69.39	87.87	86.81

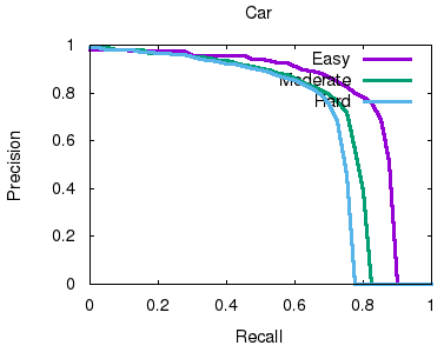
Table 4.3: Effect of varying focal loss parameters γ and α, where γ ranges from $[1, 3]$ and α is kept between $[.25, .75]$. Note that these values were chosen since they seem to achieve better performance as reported in [47].

			AP_{3D} (%)			AP_{BEV}(%)			AP_{2D}(%)		
Method	Runtime (s)	Class	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
AVOD-FPN	0.1	**Car**	**81.94**	**71.88**	**66.38**	**88.53**	83.79	**77.90**	**89.99**	87.44	80.05
AVOD	**0.08**		73.59	65.78	58.38	86.80	**85.44**	77.73	89.73	**88.08**	**80.14**
AVOD-SSD	0.09		73.64	63.87	56.90	86.14	77.66	75.68	88.94	85.71	78.05

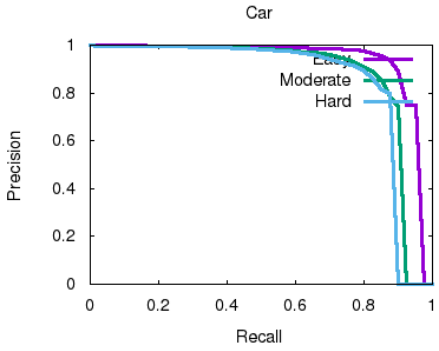
Table 4.4: AVOD-SSD results on the KITTI's evaluation server [23] compared against AVOD and AVOD-FPN, where an IoU threshold of 0.7 is used.



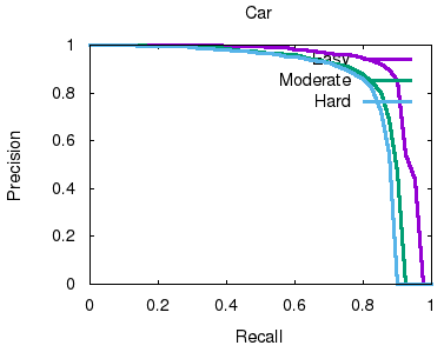
(a) AVOD-FPN 3D Car Detection.



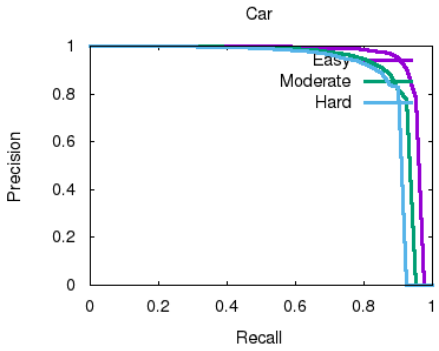
(b) AVOD-SSD 3D Car Detection.



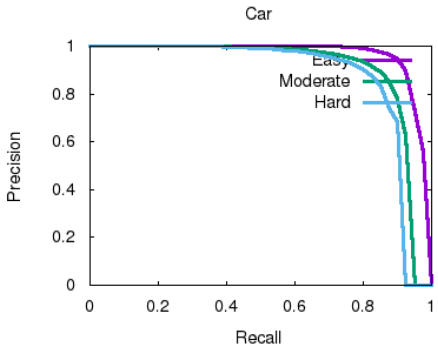
(c) AVOD-FPN Bird's eye view.



(d) AVOD-SSD Bird's eye view.

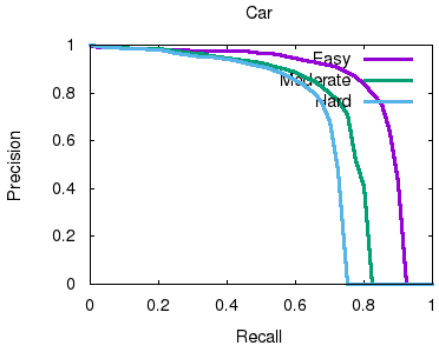


(e) AVOD-FPN 2D Car Detection.

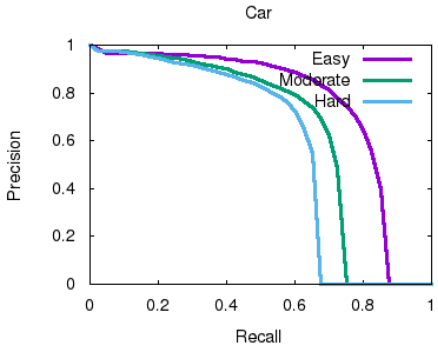


(f) AVOD-SSD 2D Car Detection.

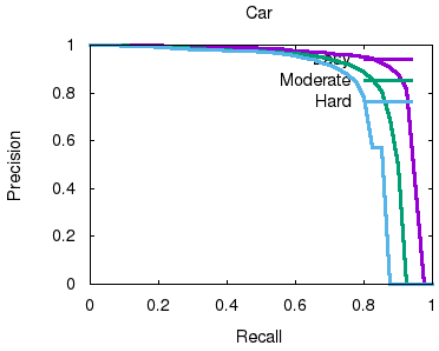
Figure 4.4: Comparison of Precision-Recall curve of AVOD-FPN vs. AVOD-SSD on the *validation* set.



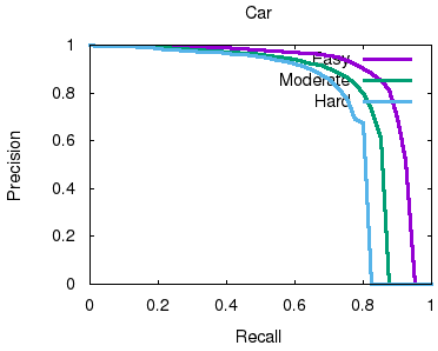
(a) AVOD-FPN 3D Car Detection.



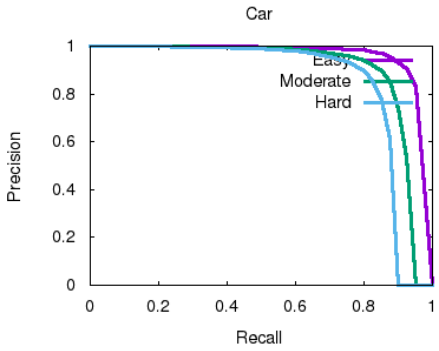
(b) AVOD-SSD 3D Car Detection.



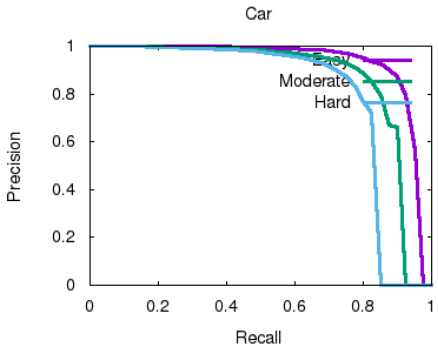
(c) AVOD-FPN Bird's eye view.



(d) AVOD-SSD Bird's eye view.



(e) AVOD-FPN 2D Car Detection.



(f) AVOD-SSD 2D Car Detection.

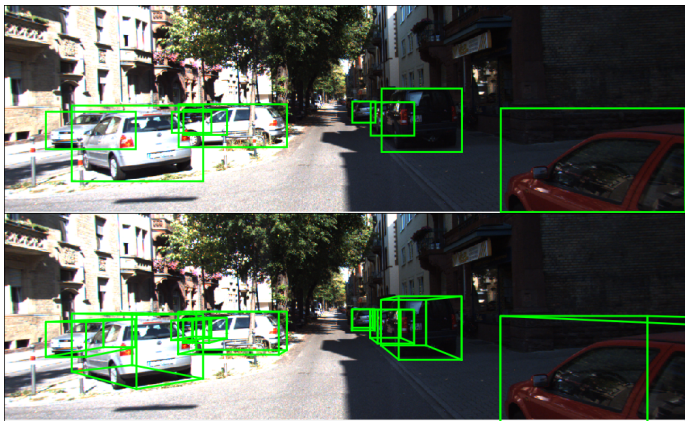
Figure 4.5: Comparison of Precision-Recall curve of AVOD-FPN vs. AVOD-SSD on the KITTI's evaluation server.

4.3.2 Qualitative Results

Figure 4.6 shows qualitative results of AVOD-SSD on KITTI's test set. It can be seen that even as a single stage detector, the network is able to detect and regress oriented bounding boxes in the scenes fairly accurately. The selected scenes include Car instances with variable orientations, occluded cases and further away instances. For these cases, AVOD-SSD can reliably detect all the objects within the scene.

Figure 4.7 shows samples where there are false detections among the detected objects by AVOD-SSD however, AVOD-FPN is more robust towards false detections. Note that such false detections tend to occur within sections where the BEV map feature might represent edges and lines similar to BEV features representing an object such as a Car. Such instances might be confusing, and without the second stage refinement to identify these detections, they can lead to false detections. It must be noted that although the network's confidence score of such false detections are typically low (sometimes as low as 0.1), we typically allow detections even with low confidence score in order to achieve high recall. The overall accuracy will depend on a good balance between high recall and high precision.

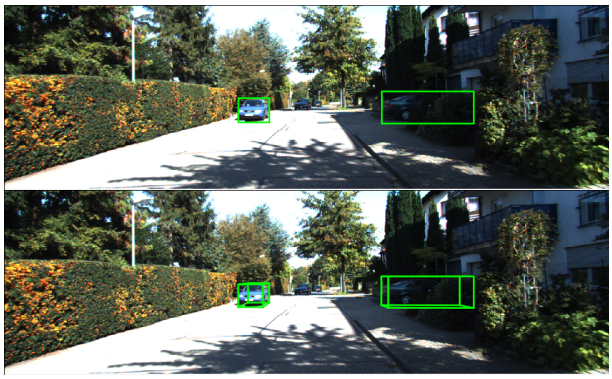
(a) AVOD-SSD predictions on sample 000013



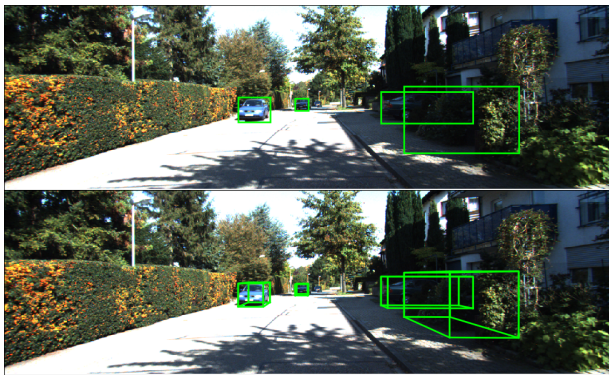
(b) AVOD-SSD predictions on sample 000030

(c) AVOD-SSD predictions on sample 000102

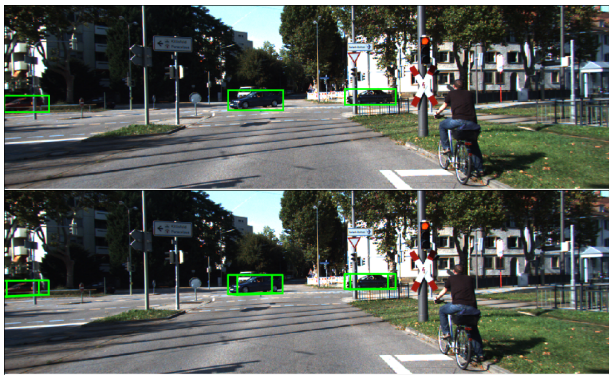
Figure 4.6: AVOD-SSD qualitative results on the KITTI test set.



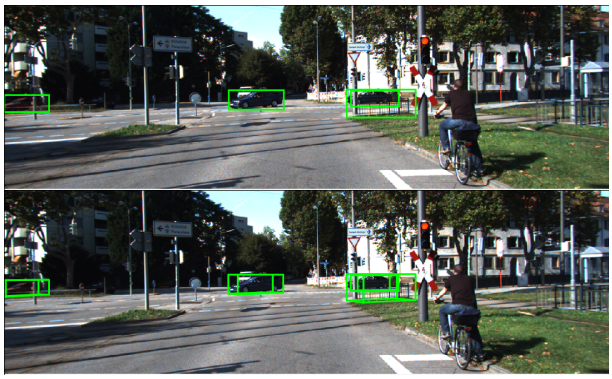
(a) AVOD predictions on sample 000203



(b) AVOD-SSD predictions on sample 000203



(c) AVOD predictions on sample 000245



(d) AVOD-SSD predictions on sample 000245

Figure 4.7: AVOD-FPN vs AVOD-SSD qualitative comparisons showing FP Car instances detected by AVOD-SSD on the KITTI test set.

Chapter 5

Conclusion

This work introduced AVOD, a 3D object detector for autonomous driving scenarios. The AVOD architecture is differentiated from the state-of-the-art by using a high resolution feature extractor coupled with a multimodal fusion RPN architecture. AVOD in turn is able to produce accurate region proposals for small classes in road scenes. Experiments on the KITTI dataset showed the performance boost of AVOD over the state-of-the-art on the 3D localization, orientation estimation, and category classification tasks. AVOD-SSD is introduced as a single stage 3D detector, and this work shows how to train a 3D single stage detector that can match the accuracy of a two-stage detector. Finally the accuracy and speed performance of AVOD vs. AVOD-SSD are studied.

5.1 Limitations

5.1.1 2D representation of 3D data

Although the accuracy performance of AVOD compared to other networks that process LIDAR points directly, is close, AVOD still relies on BEV view. F-PointNet outperforms AVOD on the task of pedestrian and cyclists detection. This could be due to two reasons, the fact that F-PointNet uses a pre-trained 2D detector which can more reliably detect smaller objects such as pedestrians, and the fact that raw 3D data has much richer information compared to BEV view. Ideally, a more suitable technique should fuse all the available information such as image, BEV and the point cloud data directly. However, this will increase the processing time significantly. Networks such as F-PointNet can still

achieve reasonable speed-time because they rely on the results of a 2D detector which generates a small number of candidate proposals compared to the RPN and hence reduces the processing time by a significant amount. These methods also require a pre-trained 2D network to be able to perform reliably.

5.1.2 Reliance on the Estimated Ground-plane

AVOD relies on an estimated ground-plane in order to generate and place the anchors on the grid. Although this works fairly well on the KITTI dataset, this fails in the case where the ground-plane estimation is inaccurate due to hills in the environment and other variations where the ground-plane is highly variable. In contrast, methods such as F-PointNet or other similar networks that rely on 2D detection, completely remove the need for ground-plane estimation.

5.2 Future Work

5.2.1 Speed Optimization

Both AVOD and AVOD-SSD networks can be made faster by optimizing certain operations. Some pre-processing operations could be moved onto the GPU to run faster, such as BEV generation. This work compared the performance of a 3D single stage detector to a two-stage detector; however, the speed boost achieved by single stage 2D detectors did not transfer well due to the feature extractor layers and high number of potential anchors to process. A different method for feature extraction as well as generating anchor proposals can be studied.

5.2.2 Processing Sequential Information

Most current 3D detectors process single frames only, one reason being the computational overhead of processing 3D information. However in the case of autonomous driving, a lot of information can be gathered via sequential frames. Information from previous frame can also be injected into the network. One possible future work is to explore the application of Recurrent Neural Networks to process video sequences. Another possible area to explore is a combined pipeline for object detection and tracking where tracking information can also guide object detection and vice versa.

References

[1] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *European Conference on Computer Vision (ECCV)*, pages 354–370. Springer, 2016.

[2] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Céline Teulière, and Thierry Chateau. Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2147–2156, 2016.

[4] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[5] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals using stereo imagery for accurate object class detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017.

[6] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems (NIPS)*, pages 379–387, 2016.

[8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893 vol. 1, June 2005.

[9] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, June 2009.

[10] Piotr Dollar, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. In *Proceedings of the British Machine Vision Conference*, pages 91.1–91.11. BMVA Press, 2009. doi:10.5244/C.23.91.

[11] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[12] Leonardo Araujo dos Santos. Artificial intelligence gitbook. https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html.

[13] NVIDIA DRIVE PX: Scalable AI Supercomputer For Autonomous Driving. https://www.nvidia.com/en-us/self-driving-cars/drive-px/.

[14] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, March 2016.

[15] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1355–1361. IEEE, 2017.

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

[17] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[18] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[19] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[20] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, Jun 2010.

[21] Pedro F Felzenszwalb, Ross B Girshick, and David McAllester. Cascade object detection with deformable part models. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 2241–2248. IEEE, 2010.

[22] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[23] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361. IEEE, 2012.

[24] Michael Giering, Vivek Venugopalan, and Kishore Reddy. Multi-modal sensor registration for vehicle perception via deep neural networks. In *High Performance Extreme Computing Conference (HPEC), 2015 IEEE*, pages 1–6. IEEE, 2015.

[25] R. Girshick. Fast R-CNN: towards real-time object detection with region proposal networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, Dec 2015.

[26] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, June 2014.

[27] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

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

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

[30] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *European Conference on Computer Vision (ECCV)*, pages 345–360. Springer, 2014.

[31] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

[32] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (CVPR)*, 37(9):1904–1916, Sept 2015.

[33] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016.

[35] 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. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[36] Lichao Huang, Yi Yang, Yafeng Deng, and Yinan Yu. Densebox: Unifying landmark localization with end to end object detection. *arXiv preprint arXiv:1509.04874*, 2015.

[37] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[39] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou,

and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105. Curran Associates, Inc., 2012.

[41] Jean Lahoud and Bernard Ghanem. 2d-driven 3d object detection in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4622–4630, 2017.

[42] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Dec 1989.

[43] Bo Li. 3d fully convolutional network for vehicle detection in point cloud. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[44] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3d lidar using fully convolutional network. In *Proceedings of Robotics: Science and Systems*, AnnArbor, Michigan, June 2016.

[45] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *Proceedings. International Conference on Image Processing*, volume 1, pages I–900–I–903 vol.1, 2002.

[46] T. Y. Lin, P. Dollr, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, July 2017.

[47] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

[48] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *European Conference on Computer Vision (ECCV)*, pages 21–37. Springer, 2016.

[49] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999.

[50] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[51] Constantine P Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. In *Computer vision, 1998. sixth international conference on*, pages 555–562. IEEE, 1998.

[52] Pedro O. Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 1990–1998, Cambridge, MA, USA, 2015. MIT Press.

[53] Cristiano Premebida, Joao Carreira, Jorge Batista, and Urbano Nunes. Pedestrian detection combining rgb and dense lidar data. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4112–4117. IEEE, 2014.

[54] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. *arXiv preprint arXiv:1711.08488*, 2017.

[55] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[56] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[57] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016.

[58] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, July 2017.

[59] Jimmy Ren, Xiaohao Chen, Jianbo Liu, Wenxiu Sun, Jiahao Pang, Qiong Yan, Yu-Wing Tai, and Li Xu. Accurate single stage detector using recurrent rolling convolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[60] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence,

D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 91–99. Curran Associates, Inc., 2015.

[61] Zhile Ren and Erik B Sudderth. Three-dimensional object detection and layout prediction using clouds of oriented gradients. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1525–1533, 2016.

[62] 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 (IJCV)*, 115(3):211–252, 2015.

[63] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[64] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 761–769, 2016.

[65] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[66] Shuran Song and Jianxiong Xiao. Sliding shapes for 3d object detection in depth images. In *European Conference on Computer Vision (ECCV)*, pages 634–651. Springer, 2014.

[67] Shuran Song and Jianxiong Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 808–816, 2016.

[68] Kah Kay Sung. *Learning and Example Selection for Object and Pattern Detection*. PhD thesis, Cambridge, MA, USA, 1996. AAI0800657.

[69] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, and Sergey Ioffe. Scalable, high-quality object detection. *arXiv preprint arXiv:1412.1441*, 2014.

[70] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[71] R. Vaillant, C. Monrocq, and Y. Le Cun. An original approach for the localization of objects in images. In *1993 Third International Conference on Artificial Neural Networks*, pages 26–30, May 1993.

[72] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, 115(1):1–28, 2015.

[73] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems*, 2015.

[74] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. Pointfusion: Deep sensor fusion for 3d bounding box estimation. *arXiv preprint arXiv:1711.10871*, 2017.

[75] Koichiro Yamaguchi, David McAllester, and Raquel Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *European Conference on Computer Vision (ECCV)*, pages 756–771. Springer, 2014.

[76] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *arXiv preprint arXiv:1711.06396*, 2017.