# SPARSE VOXEL BASED 3D OBJECT DETECTION FROM RGB-D DATA

A Thesis

by

Eren Balatkan

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
January 2021

# SPARSE VOXEL BASED 3D OBJECT DETECTION FROM RGB-D DATA

Approved by:

---

Assist. Professor M. Furkan Kıraç, Advisor
Department of Computer Science
*Özyeğin University*

---

Professor Erhan Öztop
Department of Computer Science
*Özyeğin University*

---

Assist. Professor Emre Uğur
Department of Computer Engineering
*Boğaziçi University*

Date Approved: 18 January 2021

*To My Grandfather*

# ABSTRACT

Accurate and fast 3D object detection plays a role of paramount importance for safe and capable autonomous machines. LiDAR point cloud based methods have demonstrated impressive results, yet expensive LiDAR sensors make such approaches infeasible for wide-scale adaptation. Camera based methods on the other hand are performing sub-optimally given safety and accuracy requirements. Traditionally, camera based 3D object detection is performed by generating pseudo-LiDAR point clouds from RGB-D data and using point-cloud based methods, however, irregular nature of point cloud data representation makes it challenging to exploit spatial local correlations on 3D space and point cloud based methods generally suffer from this. We propose Sparse Voxel based 3D Object Detection, our approach differs from traditional approaches by converting point cloud information to sparse voxel grid and utilizing sub-manifold sparse convolutions to extract information instead of PointNet based models. Our approach not only outperforms its point-cloud based counterparts with a wide margin but also comes with the advantage of being efficient to compute.

# ÖZETÇE

Doğru ve hızlı 3B nesne algılama, güvenli ve başarılı otonom makineler için çok önemli bir rol oynuyor. LiDAR nokta bulutu tabanlı yöntemler etkileyici sonuçlar östermesine rağmen, pahalı LiDAR sensörleri bu tür yaklaşımları geniş ölçekli adaptasyon için imkansız hale getiriyor. Öte yandan kamera tabanlı yöntemler, güvenlik ve doğruluk gereksinimleri altında yetersiz kalıyor. Geleneksel olarak, kamera tabanlı 3B nesne algılama, RGB-D verilerinden LiDAR nokta bulutları oluşturarak ve nokta bulutu tabanlı yöntemler kullanılarak gerçekleştirilir, ancak nokta bulutu veri temsilinin düzensiz doğası, 3B üzerinde uzamsal yerel bağıntılardan yararlanmayı zorlaştırır. uzay ve nokta bulutu tabanlı yöntemler genellikle bundan muzdariptir. Bu amaçla, Seyrek Voxel tabanlı 3B Nesne Algılamasını öneriyoruz, yaklaşımımız, nokta bulutu bilgilerini seyrek voksel gösterimine dönüştürerek ve PointNet tabanlı modeller yerine bilgi elde etmek için alt-katmanlı seyrek konvolüsyonları kullanarak geleneksel yaklaşımlardan farklılaşmaktadır. Yaklaşımımız yalnızca nokta bulutu tabanlı muadillerini geniş bir marjla geride bırakmakla kalmıyor, aynı zamanda hesaplama için verimli olma avantajıyla birlikte geliyor.

# ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Assist. Prof. Dr. M. Furkan Kıraç. Dr. Kıraç is a man of knowledge and an expert coder. It was through Dr. Kıraç's exceptionally well designed courses I have learned object oriented programming, became proficient with modern C++ and built strong foundations on computer vision with which I was able to complete this thesis. Knowing that I had an advisor of such proficiency gave me confidence to tackle a challenging problem and emotional support needed to overcome it.

I thank Prof. Dr. Erhan Öztop and Assist. Prof. Dr. Emre Uğur for their serving on my thesis commitee.

I would like to thank my friends at Özyeğin University; Melih Arıcı, Uluç Şahin, Berkay Bayram, Ekrem Çetinkaya, Ammar Raşid, Yakup Görür, Emir Arditi, Umut Çakan, Begüm Sunal and Cihan Eran.

Finally, I would like to thank my family, my mother Nurdan for her continual support throughout my life, my father İbrahim for encouraging me to always aim higher, my aunts Selma and Saadet and my uncles Görkem and Hüseyin for being there whenever I may need them.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

## *1.1 Motivation*

With advancing technology and developments in artificial intelligence based control systems, autonomous cyber-physical systems, such as autonomous cars and robots are becoming more common. Accurate and efficient vision systems for 3D object detection are crucial for such machines. LiDAR based vision systems are accurate but costs of LiDAR sensors limit their applicability. Camera based solutions are much more feasible but performance of camera based approaches to 3D object detection are not yet high enough to ensure safety of such systems. One of the main challenges involved in camera based 3D object detection is the lack of explicit spatial information in images captured by cameras. Various stereo and monocular methods has been proposed for estimating depth information from images, but estimated depth maps are noisy and methods utilizing such estimated depth maps must be robust against noise. Despite the challenges, the importance of this problem attracts attention nonetheless.

Most of the early work done on monocular 3D object detection operates by lifting 2D bounding boxes or key-points to 3D space in order to estimate 3D bounding box parameters. Recent improvements in monocular depth estimation [1, 2], allows for estimation of accurate depth maps and has opened the way for point cloud based approaches to tackle this problem. Recent work generates pseudo-LiDAR point clouds from RGB images by utilizing a monocular depth estimator and use PointNet[3] based methods for extracting features from pseudo-LiDAR point cloud. Because of the irregular nature of point clouds, analysis of spatial local patterns in data is difficult and PointNet based methods tend to suffer from this issue, leading to sub-optimal

performance.

In order to better analyze spatial local patterns compared to PointNet based solutions, in this work, we propose to convert pseudo-LiDAR point clouds to sparse voxel grids and use a sub-manifold sparse convolutional neural network architecture to extract features. Proposed approach, compared to its PointNet based counterparts, achieves over double the average precision while also being efficient. We further propose Bin-Mixing layers. By representing the task of regression as a bin-weighing problem, Bin-Mixing layers reduce the difficulty of regression, resulting in a better fit into data and robustness against overfitting.

Our contributions in this thesis can be summarized as three-fold. (1) S3D2 sub-manifold sparse convolutional network architecture for 3D information extraction that is significantly more accurate and relatively fast compared to PointNet based architectures. (2) Bin-Mixing layer that simplifies the task of regression by representing regression as a bin-weighing problem, resulting in improvements in convergence times, robustness against overfitting and accuracy. (3) Simple, yet effective pipeline for RGB-D object detection that takes advantage of S3D2 architecture and Bin-Mixing layer, SparVox3D.

## *1.2 Thesis Outline*

Rest of this thesis is organized as follows:

Chapter 2.1 gives information about neurons, activation functions, types of neural networks, a commonly used regularization technique known as Dropout, common problems encountered in neural networks and methods for tuning the weights of the neural networks.

Chapter 2.3 gives overview of some of the popular methods for analyzing 3D information, ie. PointNet, PointNet++[4] and Sub-manifold Sparse Convolutional Networks[5].

Chapter 2.4 makes brief information of some of the tasks that were utilized in this thesis.

Chapter 3 gives information about other approaches to monocular 3D object detection.

Chapter 4 contains detailed information about our approach, S3D2 Module, Bin-Mixing layer and our overall pipeline.

# CHAPTER II

# BACKGROUND

## *2.1 Neural Networks*

### 2.1.1 Neuron

Neurons are the basic building blocks of neural networks. Inspired by their biological counterparts, artificial neurons are designed to activate when they encounter a specific pattern in their inputs. Although each neuron calculates a simple function over its inputs, millions of neurons combined together can learn to recognize extremely complex patterns. Simplicity of their design makes scaling neural networks and debugging them relatively simple.

Each neuron takes arbitrary inputs $x_1, x_2, x_3, ..., x_n$ and has weights $w_1, w_2, w_3, ..., w_n$. Activation of a neuron is calculated by taking sum product of a neurons inputs and its weights and passing the result to an non-linear activation function $\sigma$. Figure 2.1.1 gives the mathematical formulation of a neuron.

### 2.1.2 Activation Functions

A linear neuron will apply only a linear transformation to its outputs. And multiple linear transformations stacked on top of each other still end up being equal to single linear transformation just with different parameters. Given the complexity and the difficulty of the problems that machine learning seeks to solve, linear transformations are not feasible. Non-linear activation functions are designed to eliminate this issue. As non-linear transformations stacked on top of each other make the over all transformation more complex with each transformation. Which allows neural networks to become universal function approximators as long as sufficiently many neurons and layers are available[6].

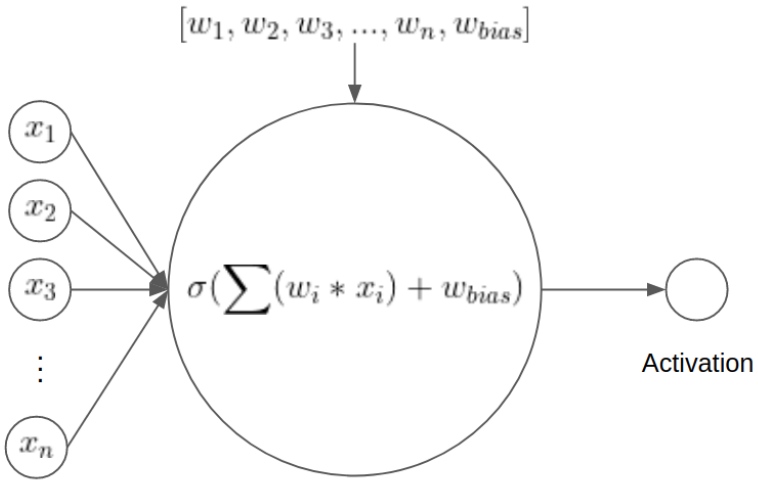


**Figure 1:** Visualization of a single neuron.

Different activation functions have different properties in terms of efficiency of their calculations, expressiveness of neurons and how much they suffer from vanishing or exploding gradients problem. Many different activation functions have been used in neural networks. Yet ReLU and its variations has become industry standard that is extremely commonly used in both research and practice alike. Main advantage of ReLU is that it is extremely efficient and that it doesn't suffer from vanishing or exploding gradients problem. ReLU however, comes with risk of having dead neurons if activation values are frequently below 0. However, this is easily fixed on variations of ReLU such as Leaky ReLU[7].

### 2.1.3 Feed-forward neural network

These can be considered as one of the most simple forms of neural networks. Feed-forward neural networks consists of layers where each neuron in layer $n+1$ is connected to every neuron in layer $n$, There are multiple drawbacks to this design, expensive computation, vast number of parameters and it suffers from vanishing and exploding gradients problems.

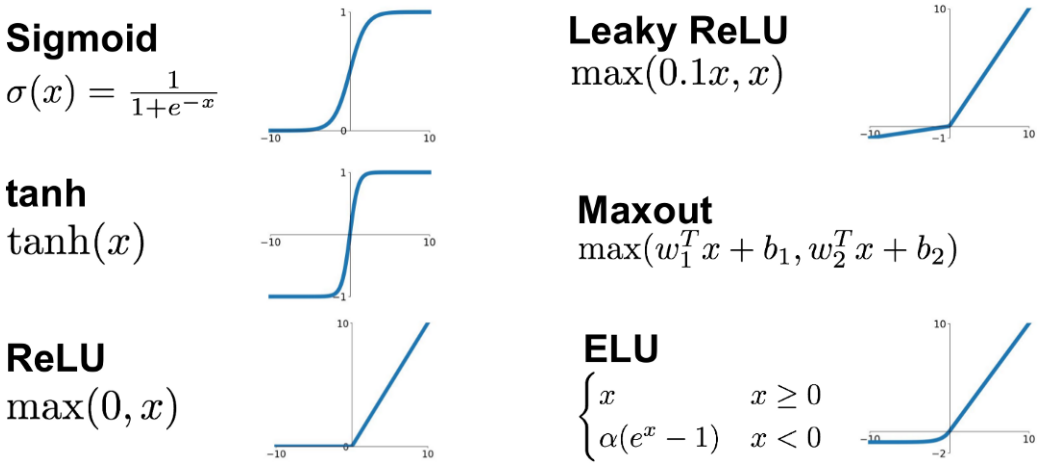


**Figure 2:** Visualizations of various activation functions.
Image source: [8]

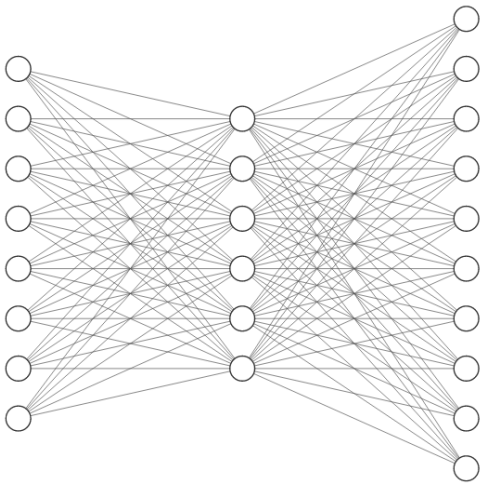

**Figure 3:** Visualization of a simple feed-forward neural network with 3 layers.
Figure was generated with [9].

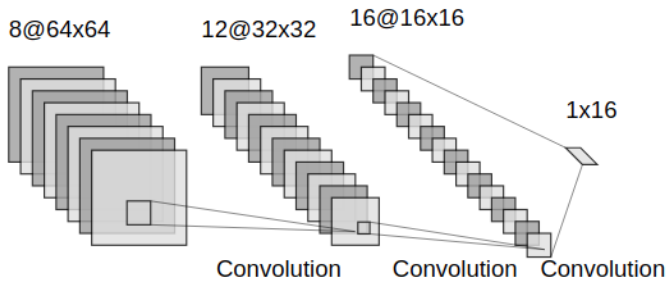


**Figure 4:** Illustration of a convolutional neural network.
Figure was generated with [9].

### 2.1.4 Convolutional Neural Networks

Convolutional filters have proven themselves as extremely capable, yet efficient tools for extracting features in computer vision tasks. CNN's, driven by the success of traditional convolutional filters, replicate this operation in efficient and learnable way in deep learning. Convolutions are extremely powerful, almost all forms of modern neural networks take advantage of convolutional layers one way or another, not only on vision related tasks, but also in tasks such as reinforcement learning and natural language processing.

Convolutional filters can be summarized as learnable NxM matrices that we slide over inputs. By padding the input signal, or sliding the convolution with different stride radios we can have control over the shape of the output signal. Primary advantage of convolutions is that they are extremely good at analyzing spatial local correlations in data. Where each convolution searches for a pattern in signal and gets high activation when it encounters that pattern.

Main disadvantage of convolutional neural networks is that they tend to have limited field of view. Various techniques, such as attention based modes and dilated

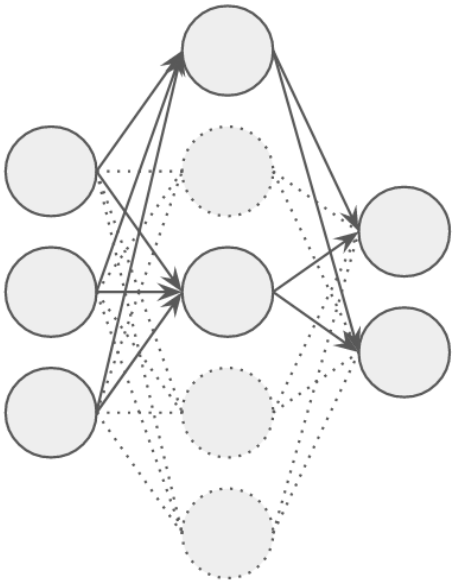

**Figure 5:** Illustration of Dropout Regularization during training.

convolutions have been proposed to combat with the field of view problem.

### 2.1.5 Dropout Regularization

Neural networks are extremely capable learners. This capacity can often manifest as a disadvantage on difficult problems with small datasets, in which case, neural networks tend to learn to become a hash map that recognizes inputs and outputs labels from memory instead of learning the actual functional logic between the inputs and outputs.

Dropout[10] strategy has been proposed to combat with this issue. Dropout works by randomly disabling some of the neurons during training. This ensures that subsequent neurons will receive different inputs even on same training samples, which makes learning of hash-map style mappings harder and this also forces neural network to have multiple neurons that are looking for same patterns since it is important that most beneficial features propagate and not get dropped out.

During inference, randomly dropping out neurons is risky and harms overall performance. But since applying dropout changes the mean of the outputs, to ensure we have same scale on inference where no neurons will be dropped, we scale outputs of neurons by a factor of $p$ where $p$ is the ratio of dropped neurons.

### 2.1.6 Common problems in Neural Networks

Although extremely capable learners, neural networks suffer from some problems that researchers and engineers must be aware of. In this section, I will cover some of the most common such problems.

#### *2.1.6.1 Vanishing Gradients*

At their core, neural network layers are batch matrix multiplications followed by non-linearities. Multiplying continuously with a value between $-1$ and 1 will, in a long neural network, result in gradients approaching 0 and hinder the learning process. Architectures such as ResNets[11] and DenseNets[12] are designed to introduce skip connections that alleviate this problem somewhat.

In this case, activations are also important, gradients of the activation functions such as Sigmoid or TanH also approach 0 as their inputs go to $\pm\infty$. ReLU on the other hand, has a constant derivative of 1 as long as input is above 1 which makes it much more robust against vanishing gradients problem, allowing for training of deeper networks.

#### *2.1.6.2 Exploding Gradients*

Exploding gradients problem can be seen as opposite of vanishing gradients, this happens when we make repeated multiplications with values above 1 or lower than $-1$. That being said, exploding gradients problem is rarer than vanishing gradients problem. This is most frequently seen on recurrent neural networks or in situations with unstable training procedures such as deep reinforcement learning or generative

adversarial networks. Exploding gradients problem can be effectively combated using gradient clipping.

#### *2.1.6.3 Underfitting and Overfitting*

Underfitting happens when either regularization is too aggressive or models complexity is lower than the complexity of the generative function of the data. Overfitting happens when model is more complex than generative function of data. In situations where underfitting happens, one must either use less aggressive regularization or increase model complexity. When overfitting happens, one must employ more aggressive regularization, decrease model complexity or use stronger preprocessing.

### 2.1.7 Gradient Descent

Given weights $w$, data $D$ and loss function $L$, our goal is to find the $w$ that minimizes $L$ on dataset $D$. Given the size of modern datasets and neural networks, calculating global minimum is far from being computationally feasible. Computational limitations that prevent us from being able to calculate global minimum gave birth to new set of techniques that try to fit onto a local minimum. Although many algorithms have been proposed for tuning the weights of neural networks, gradient descent remains as the superior and by-far the most popular technique.

Gradient descent is a technique for searching local minima, by calculating the derivative of $L$ with respect to $w$ on a small sub-set of data, we can calculate the direction in which we must chance the weights in order to decrease the loss. By repeating this algorithm many times on different subsets of data, we will guarantee on a local minima. While local minima may not be the optimal solution, for most problems it works well enough and different optimization techniques for gradient descent further help us reach a better local minimum.

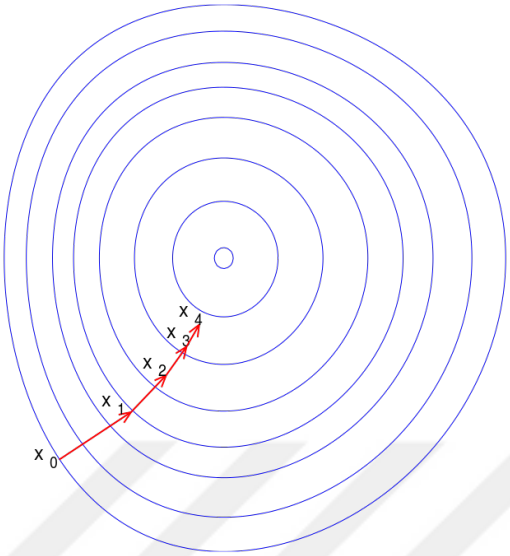


**Figure 6:** Illustration of gradient descent steps.
Image source: [13]

### 2.1.8 Optimizers

Optimizers can be seen as slight modifications to gradient descent algorithm that are done in order to make convergence faster and help network converge at a better local minima.

#### *2.1.8.1 Adam Optimizer*

Adam(Adaptive Moment Estimation) optimizer[14] is the de-facto standard in industry. Based on work done by RMSProp[15] and AdaGrad[16]. Adam works by changing learning rates on per-parameter basis. By ensuring parameters with small gradients have higher learning rate and parameters with big gradients have smaller learning rates, it makes training robust against instabilities but also ensures fast convergence. To prevent getting stuck on local minima Adam by adapting momentum strategy. Instead of calculating update step based only on current batch of inputs, by calculating it based on current samples averaging it with previous updates with exponential decay, momentum strategy helps us to not get stuck on small local minimas.

**Figure 7:** Rendering of pseudo-LiDAR point cloud.
Point cloud is generated from RGB image using monocular depth estimator.

## *2.2 3D Data Representations*

### 2.2.1 Point cloud representation

Point cloud representation consists of points where each point has 3D coordinates and may optionally include additional features. Point-Cloud information is generally kept in a unordered, permutation invariant list. This method of keeping information is irregular and location based queries require iteration over entire dataset. Irregularity is further reinforced by the fact that points only have central coordinates in 3D space and no size.

### 2.2.2 Sparse Voxel Representation

Sparse voxel representation usually consists of two lists, first list contains features of every voxel, and second list contains information about the location of every voxel. These two lists are ordered in such way that element $i$ at both lists contain information about same voxel. Sparse voxel representation is regular, every voxel is of same size and neighbouring voxels are always $x$ distance apart where $x$ is the length of single

**Figure 8:** Rendering of sparse voxel information.

corner of voxel. It is possible to construct a hash map from this data where locations list makes the keys of the hash map and features list makes for the values of the hash map. Due to regularity of the sparse voxel representation it is possible to make $O(1)$ look ups for neighbouring voxels of a given voxel.

## *2.3 3D Information Processing*

Due to curse of dimensionality, as we move on from 2D representations to 3D representations, sparsity increases significantly. Because of added sparsity, utilization of techniques that were developed for 2D on 3D setting becomes significantly harder. And added dimension means that techniques that do not account for sparsity will come at a significant computational cost. Multiple new techniques have been proposed for extracting information from 3D data, in this section we will cover some of the more popular ones.

### 2.3.1 PointNet

PointNet[3] is one of the first architectures for deep learning based point-cloud processing. PointNet operates by calculating a linear function over every point individually and then samples most important points using max-pooling operation. Due to the fact that point clouds are invariant to permutations in data representation, operation used for sampling must be order invariant, making max-pool an ideal function for such operation. Afterwards, samples obtained through max pooling operation are aggregated to global feature descriptor. For classification tasks this descriptor can be used for prediction, for segmentation tasks, features of this descriptor are concatenated back to points so that features of each point also give description of global context. Straight-forward architecture of PointNet makes it extremely efficient for parsing of point clouds. Such efficiency can make it ideal choice based on the requirements of the problem.

Architecture of PointNet makes extracting patterns that consists of multiple-points extremely difficult. PointNet cannot automatically detect occurrences of same pattern in different partitions of data automatically. At every aggregation step, information of only a single point is aggregated globally, this means that for analyzing occurrence of a pattern that consists of 5 points in 3 different subspaces of data requires 15 aggregation steps. Making PointNet extremely inefficient for extraction of patterns.

### 2.3.2 PointNet++

PointNet++[4] is an improvement over PointNet. PointNet++ aims to better take advantage of local patterns in data. By applying PointNet recursively on nested partitions of the point cloud, PointNet++ feature descriptors now encode local contexts instead of single global context. However, PointNet++ still comes with some limitations. Different groupings of points can result in failure to analyze local patterns

properly, selecting which points to group are one of the major challenges. Max-pool based feature aggregation means that at every feature aggregation step, only one point can be appended, which makes analysis of patterns that are the result of many points very challenging for this architecture. And nesting of points at every step is a computationally expensive operation which makes this architecture inefficient. Compared to PointNet, PointNet++ is significantly more computationally intensive, which results in loss of one of the main advantages of PointNet. However, PointNet++ can detect multiple occurrences of the same pattern in different subspaces easily as it is applied to nested partitions of the data separately.

### 2.3.3 Sub-manifold Sparse Convolution

Compared to 2D data, 3D data tends to contain significantly higher amounts of sparsity due to the curse of dimensionality. Traditional convolutions, when applied to highly sparse 3D data, tend to perform underwhelming. Applying traditional convolutions to voxel maps bring two major issues that prevent effective learning: dilation of output feature maps that prevent effective propagation of actual information as number of layers increase and in-ability to gain any speed ups from the sparsity of the data, which becomes more important as number of dimensions increase and curse of dimensionality becomes more prominent.

Sub-manifold sparse convolutions[5] are designed to effectively combat with both of these problems. Unlike traditional convolutions that simply calculate same convolution operation over whole input space without any regard for sparsity of the data, sub-manifold sparse convolutions only calculate when an area is considered an active state. Active states are states where the input corresponding to the center of the convolutional filter is not sparse. This effectively combats with the dilation of the output space. Sparse parts of the input space are replaced with 0's so that they do

not affect the outcome of the convolution operation. With these modifications, submanifold sparse convolutions not only allow for effective learning on sparse 3D data, but also bring significant efficiency gains on data with high sparsity.

## *2.4 Related Tasks*

### 2.4.1 Instance Segmentation

Instance segmentation can be seen as a more advanced form of object detection. In object detection our goal is to estimate 2D object bounding box per object in image plane. In instance segmentation, instead of estimating just a bounding box for the object, we detect the pixels that belong to the object instead by estimating an instance mask. Estimated instance mask is of same height and width as image but only has 1 channel.

### 2.4.2 Monocular Depth Estimation

Monocular depth estimation refers to the task of estimating per-pixel depth values in meters from a single image. Lack of explicit spatial information on images makes this problem extremely challenging. Furthermore, this problem is ambiguous, a toy car with model that matches the original but located close to camera, when shooter through a camera, will look same as original car that is meters away from camera.

### 2.4.3 3D Object Detection

3D object detection can be seen as an extension to 2D object detection across depth plane. In this task, instead of estimating 2D bounding box parameters in image plane, goal is to detect 3D bounding box parameters relative to the camera.

# CHAPTER III

# RELATED WORK

## *3.1 Image based Approaches*

**FQNet[17]** unlike other 2D based approaches, start by regressing dimension and regression of the object with an anchor-based method. Afterwards, they sample large number of candidate 3D object bounding box'es. They propose FQNet, which predicts 3D IoU between candidate 3D bounding boxes and the object based only on 2D information. Using FQNet, they select the best candidate as their prediction of 3D bounding box.

**ROI-10D[18]** lifts 2D detections to 3D using differentiable ROI lifting layers and a monocular depth estimator. Instead of optimizing these separately, by lifting detections to 3D and calculating these quantities on 3D space they jointly optimize metric misalignment.

**GS3D[19]** use off-the-shelf 2D detector, by utilizing well engineered techniques, they then predict a coarse cuboid based on accurate 2D detections. Afterwards, by extracting features from visible surfaces of the cuboid they then refine the coarse cuboid to get final 3D object bounding box estimations.

**3D-GCK[20]** lift 2D detections to 3D bounding boxes by predicting additional regression parameters and transforming these parameters to 3D bounding box keypoints. Since they do not use any monocular depth estimation phase, their approach is very efficient and can be employed real-time.

**RTM3D[21]** predicts 9 key points on image-plane, which correspond to projections of 3D bounding box corners and 3D bounding box center. This approach does not rely on any depth estimation or 2D object detection phase, making it even more efficient than [20] and due to high number of points giving the model robustness against noisy predictions, their results are competitive with state of the art.

**D4LCN[22]** uses depth estimator similar to point-cloud based methods, but instead of converting estimated depth map into pseudo-LiDAR point cloud, they propose a local convolutional network that automatically learns filters and receptive fields of filters from estimated depth maps. Convolutional network they propose can much more effectively reason with depth information.

## *3.2 Point Cloud based Approaches*

**Mono3D_PLiDAR[23]** is one of the first methods to utilize pseudo-LiDAR representations generated from depth maps. Similar to our work, use instance segmentator to generate point-cloud frustums that contain only points which belong to objects. They also utilize bounding box consistency module that aligns estimated 3D bounding box with estimated 2D bounding box. Unlike our work, their approach does not apply any further voxellization and instead uses PointNet based method for analyzing a sampled sub-set of the object point cloud and their pseudo-LiDAR points do not contain RGB information.

**AM3D[24]** utilize 2D bounding box detections to filter out the regions in pseudo-LiDAR point cloud that does not contain objects. They argue that directly appending RGB features to points prevent effective propagation of this additional information and instead they propose an attention based mechanism for controlling propagation of RGB features.

**RefinedMPL[25]** seek to eliminate uninformative high number of points on pseudo-LiDAR point cloud representations. They propose an unsupervised, and supervised 2D bounding box method based strategies. Their work demonstrates that it is possible to improve the performance of PointNet based approaches while using only 5% of the points in pseudo-LiDAR point cloud.

# CHAPTER IV

# METHODOLOGY

SparVox3D consists of 3 primary stages. Data Preparation, Information Extraction and Bin-Mixing stage. Goal of the Data Preparation phase is to generate object sparse voxel grid representations from RGB-D information. In Information Extraction phase, our proposed S3D2(**S**parse **3**D-**D**ense **2**D) module takes in sparse voxel grid and outputs a vector of length 128, which can be seen as summarization of 3D information. Objects that are distant from camera tend to have noisier point cloud, in order to allow network to be able to recover from such distance based noise, object point cloud center coordinate and one hot encoded object type, which is predicted by instance segmentator, is concatenated onto output of S3D2 module. After concatenation, intermediate representation becomes of length 128 + 3(object pc center coordinate) + (class count). This intermediate representation is then passed onto the Bin-Mixing module for regression of 3D object parameters. Bin-Mixing layer outputs 3D object center coordinate relative to the object point cloud center coordinate, therefore, to get the final coordinate in camera frame both of these values are added together.

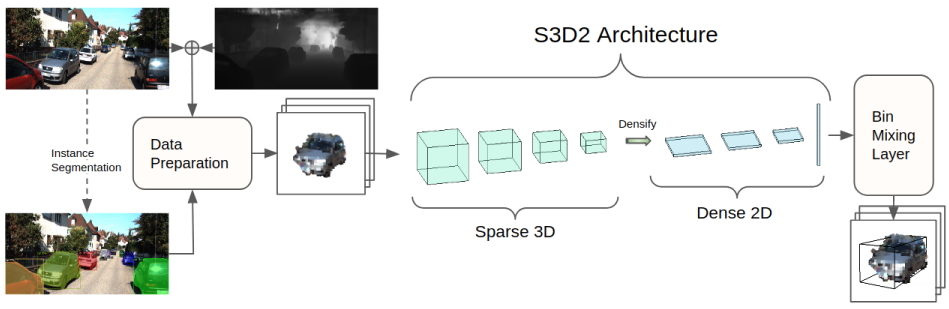


**Figure 9:** Overview of SparVox3D pipeline

Chapters 4.3, 4.4 and 4.5 contain more detailed information about our phases and figure 9 contains an overview of our framework.

## *4.1 Dataset*

We use Kitti[26] dataset for all the experiments in this thesis. Kitti dataset consists of 14.999 images and total of 80.256 labeled objects. For 3D object labels, Kitti dataset contains 7 values representing center x, center y, center z, width, height, length and heading angle. Although images and focal lengths are varying in size and focal length, variations are small and values are around $1224x370$ for images and 715 for focal length in pixels. Images are split as 7.481 images for training and validation and 7.518 images for online leaderboard comparison, whose labels ar not shared. We split training set into 3.712 and 3.769 samples for training and validation.

## *4.2 Monocular Depth Estimation*

In situations where depth information is not available, it is possible to utilize a monocular depth estimator to estimate depth maps from RGB images. We utilize DORN[1] depth estimator for our benchmarks and BTS[2] for our figures and video samples. Estimated depth maps are generally noisy. Errors made in this stage propagate onto future stages and can lead to estimation failures. One of the main weaknesses of our method is its reliance on correct depth map estimations.

## *4.3 Generating Sparse Voxel Representation*

Initially, similar to other pseudo-LiDAR point cloud based approaches, we calculate pseudo-LiDAR point clouds from RGB-D information.

Where $(i, j)$ refers to coordinate of pixel, focal refers to focal length in pixels and principal-point refers to principal point of camera, point coordinates of every pixel in RGB-D image can be calculated as follows.

**Figure 10:** Sample images from Kitti dataset

**Figure 11:** DORN depth estimator predictions on Kitti dataset.

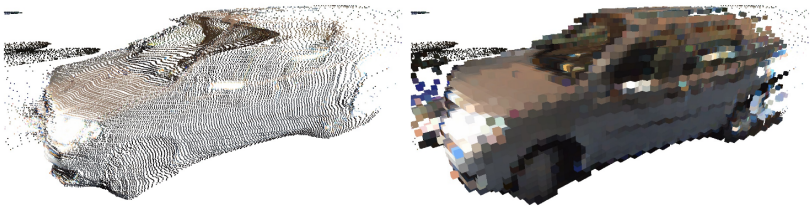

**Figure 12:** Visual comparison between point-cloud representation and sparse voxel representation of a car.
Left: pseudo-LiDAR point cloud representation, Right: sparse voxel representation.

$$
\begin{aligned}
z_{ij} &= \text{depth}_{ij} \\
x_{ij} &= z_{ij} * (i - \text{principal-point}_x)/\text{focal}_x \\
y_{ij} &= z_{ij} * (j - \text{principal-point}_y)/\text{focal}_y
\end{aligned}
\tag{1}
$$

Our approach is designed to operate in per-object fashion, that is, objects are treated as individuals regardless of whether they belong to the same image or different images. BlendMask[27] instance segmentator is utilized to detect objects on image and estimate their respective instance masks. We use AdelaiDet[28] implementation of BlendMask. Estimated instance masks are then used to filter pseudo-LiDAR point cloud so that only the points that belong to the object remain.

For each object point cloud, we first determine the center coordinates of the point cloud by finding the median point. Afterwards, we generate a sparse voxel grid of the object from points that fall within a box which is centered at the object point cloud center coordinate and has size of 6.4 in width and length and 2.4 in height. Values of voxels are determined by taking average RGB-XYZ values of points that fall within boundaries of a voxel.

## *4.4 S3D2 Network Architecture*

Traditional convolutions are proven to be exceptionally successful in 2D dense information. However, simply applying them to 3D data, which is mostly sparse, results in


**Figure 13:** Visualization of BlendMask instance segmentator predictions.

poor estimations. True information that exists within data structure gets mixed up with sparse, meaningless data. Sparse convolutions on the other hand can effectively combat this issue and bring the success of convolutions in 2D dense information to 3D sparse information. Driven by the success of their work, we design our S3D2 architecture in 2 parts. First part operates in sparse voxel space, utilizing sub-manifold sparse convolutions and sparse convolutions to downscale and collapse 3D grid in height axis into 2D sparse grid. Collapsed 2D information is highly dense, in which case, sparsity itself becomes a source of information. To take advantage of this, Dense 2D part of our S3D2 architecture is built fully using traditional convolutions.

Sparse 3D part of S3D2 architecture is composed of 4 consecutive modules, each module applies sub-manifold convolution and downscales using sparse convolutions[29] with stride 2 and padding 1. Final module makes an exception and uses stride 1 and padding 0 on sparse convolution instead. Note that, for the 2D collapse to happen as implemented, the height of the input to the final layer must be 3. Since the our sparse voxel map is of shape $[64, 24, 64]$, and each module downscales its inputs by a factor of 2, after 3 consecutive modules, input to our final module becomes of shape $[8, 3, 8]$ which when passed to final layer with stride 1 and padding 0, produces output of shape $[6, 1, 6]$ which can simply be collapsed to 2D by squeezing the second dimension. As long as height dimensions of input sparse voxel data can be expressed as $3 * 2^x$ where is a positive integer, and length and width of sparse voxel data shape is bigger than its height, we can collapse 3D sparse voxel grid to 2D grid using $x$ sparse modules followed by a final module with 0 padding.

Second part is the Dense 2D part that converts sparse 2D grid into dense 2D grid. Dense 2D part is composed of 3 convolutional layers where padding is set to 1 in all of them and stride is set to $1, 0, 0$ respectively. Finally, Dense 2D part collapses the grid from 2D into 1D using adaptive average pooling. We use Leaky ReLU[7] as the activation function for all activations.

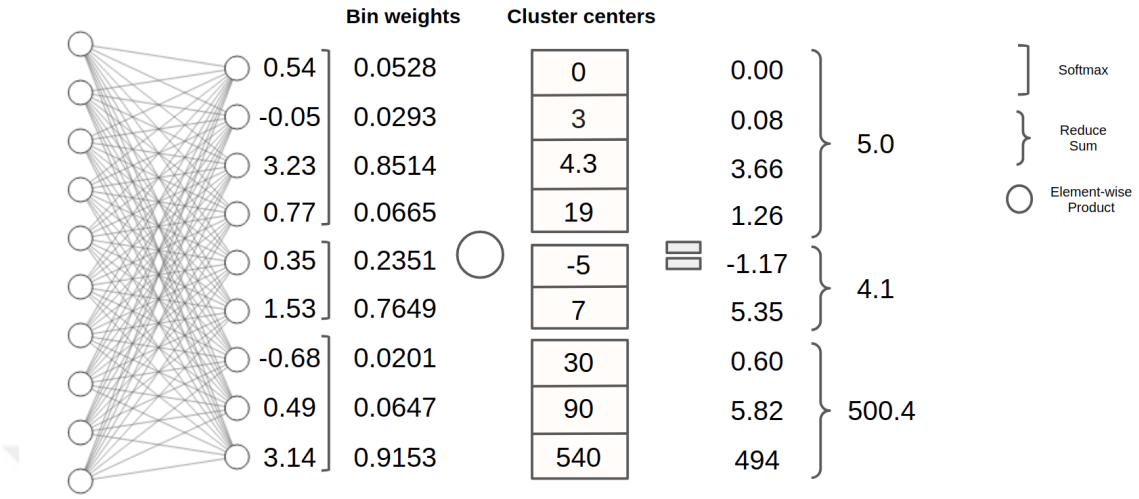


**Figure 14:** Example of our Bin-Mixing layers being utilized for regression.

## *4.5 Bin-Mixing Layer*

Opposed to their high success in voting based tasks, neural networks tend to underperform in regression based tasks, this problem becomes more prominent on small datasets as fitting too well into data results in overfit. Bin-Mixing layer is designed to effectively combat this problem by representing the task of regression as a bin-weighing problem.

Bin-Mixing layer operates by quantizing output space to "bins". Neural network then predicts weights for these bins. We then take the weighted average of bins to get our prediction for a value.

For estimating bin weights, we employ 2 layer fully connected neural network with 128 and 53 neurons respectively. Number of neurons in the last layer must be equal to the number of total bins in the Bin-Mixing layer. To further gain robustness against overfitting, we add Dropout layer between the fully connected layers with dropout probability 50%

Bin-Mixing layers require calculation of bin values before the actual training phase,

in order to get a good quantization of the output space, we utilize an unsupervised clustering algorithm, K-Means, to determine bin values. Regression parameters are assumed to be independent and cluster centers are calculated individually per-parameter. Calculated per-parameter cluster centers then serve as bin values. To ensure full data coverage, we further include max and min values in the dataset as additional bins. Value of K used in K-means is a hyper-parameter, our approach uses K of 5 for object x, y, width, height, length, 6 for angle and 10 for object z parameters.

Since Bin-Mixing layer takes a weighted average of bins to predict target value, estimated Bin-Weights must sum up to 1. In order to ensure that, we apply softmax function over Bin-Weight predictions. Afterwards, we take sum-product of softmaxed bin weights with bins. This ensures that bin-mixing layers, even though voting based, can give estimates for continuous values and are automatically differentiable.

If we are regressing more than one parameter, bins and calculated separately per-parameter and bin-weights are estimated per-parameter as well.

## *4.6 Implementation Details*

Our model is trained for 35 epochs using Adam optimizer with learning rate $1e^{-4}$ and batch size 32 using smooth L1 loss, after epoch 25 we further add corner loss multiplied by 0.1 introduced in [30]. We have found that calculating corner loss with predictions $p_1$ and $p_2$ where $p_2$ is $p_1$ rotated by $\pi$ and choosing minimum loss of these two improves stability and leads to better convergence.

Before feeding the inputs to network during training, we apply random flip to x axis with probability %50, eliminate %10 of points used for sparse voxel generation randomly, apply random rotation to object's point cloud around its center in range $[-8.5°, 8.5°]$. We add a random value in range $[0, 0.10]$ to RGB values of points and a value in range $[0, 0.05]$ to R-G-B channels separately, first addition changes overall brightness of the point cloud while second addition changes channel intensities by a

| Setting | 2D Bbox Max Height | Occlusion | Max Truncation |
|---|---|---|---|
| Easy | 40px | Fully visible | 15% |
| Moderate | 25px | Partly occluded | 30% |
| Hard | 25px | Difficult to see | 50% |

**Table 1:** Kitti Metric Categories

different value per-channel.

Running instance segmentation and depth estimation at every train step is expensive, in order to gain to make training faster, we split our training into 2 main phases, in phase 1, we run our data preparation step alone and generate object dataset that contains items where each item represents an object and contains its respective point cloud. In phase 2 we iterate over object dataset alone.

## *4.7 Metric*

KITTI Metric is used for comparisons and evaluation of our results. Estimated 3D bounding boxes with over 70% overlap over 40 recall positions are considered as match. Results are further grouped into 3 categories, Easy, Moderate and Hard based upon criteria given in table 1.

## *4.8 Results*

### 4.8.1 3D Information Extraction

| Architecture | Car 3D mAP | | | Car BEV mAP | | | Inference Speed |
|---|---|---|---|---|---|---|---|
| | Easy | Mod | Hard | Easy | Mod | Hard | |
| PointNet[3] | 2.5 | 1.7 | 1.2 | 9.5 | 6.5 | 5.2 | **232.6** |
| PointNet++[4] | 5.7 | 3.7 | 3 | 12.3 | 8.6 | 7 | 37.4 |
| S3D2(Ours) | **11** | **7.5** | **6** | **20** | **13** | **11** | 83.4 |

**Table 2:** Comparison to PointNet based architectures on Kitti validation set. Inference speeds are given in images per second

In table 4.8.1 we compare out S3D2 architecture to PointNet and PointNet++ under same architecture. Only difference in this case is that we have replaced our

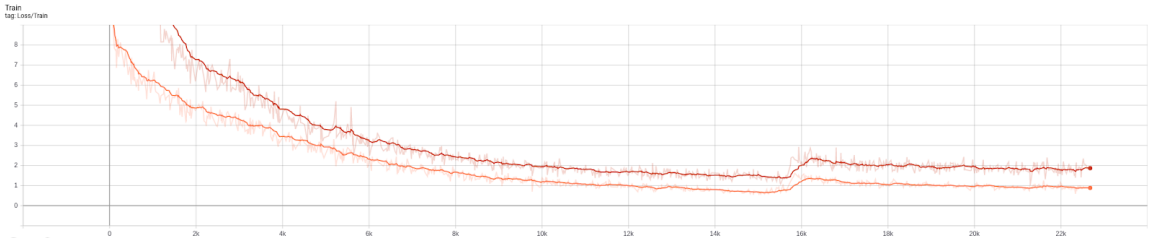

**Figure 15:** Training plot of SparVox with S3D2 module and with Bin-Mixing layer enabled and disabled.
Green plot belong represents training with Bin-Mixing layer enabled and red plot represents training with Bin-Mixing layer disabled. Loss values were smoothed with a factor of 0.9 for clearer representation.

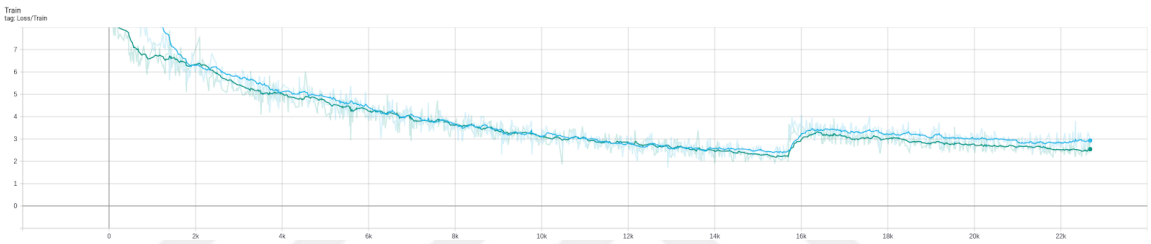

**Figure 16:** Training plot of SparVox with PointNet++ and with Bin-Mixing layer enabled and disabled.
Gray plot belong represents training with Bin-Mixing layer enabled and blue plot represents training with Bin-Mixing layer disabled. Loss values were smoothed with a factor of 0.9 for clearer representation.

information extraction phase to use PointNet or PointNet++ and modified data-preparation phase to output pseudo-LiDAR point clouds instead of sparse voxel grids.

### 4.8.2 Effects of Bin-Mixing Layers

Table 4.8.2 compares our Bin-Mixing layer to direct regression of parameters and direct regression of parameters using radial basis functions[31]. For direct regression, we modify the second fully-connected layer inside Bin-Mixing module to have 7 neurons, outputs of these neurons directly correspond to 7 regression parameters for 3D object detection instead of being bin-votes. For setting with radial basis functions, we follow the same architecture as the setting with Bin-Mixing layers but replace Bin-Mixing layer with radial basis function that takes, what was before bin-votes, as

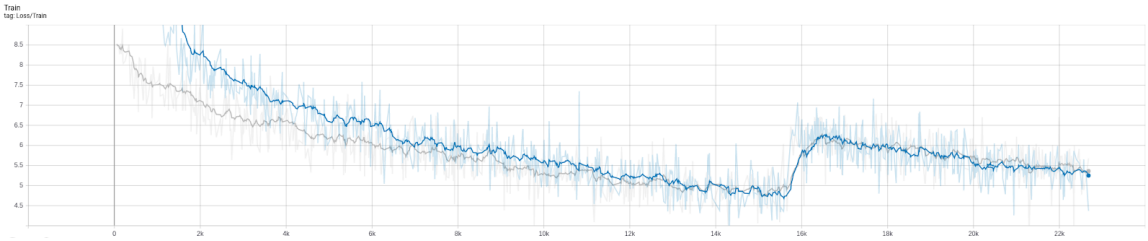


**Figure 17:** Training plot of SparVox with PointNet and with Bin-Mixing layer enabled and disabled.
Gray plot belong represents training with Bin-Mixing layer enabled and purple plot represents training with Bin-Mixing layer disabled. Loss values were smoothed with a factor of 0.9 for clearer representation.

| Architecture | Car 3D mAP | | | Car BEV mAP | | | Relative |
|---|---|---|---|---|---|---|---|
| | Easy | Mod | Hard | Easy | Mod | Hard | |
| PointNet[3] Reg | 0.10 | 0.11 | 0.12 | 0.76 | 0.71 | 0.58 | - |
| PointNet[3] RBF | 1.1 | 0.8 | 0.6 | 4.5 | 3.1 | 2.6 | 1000% |
| PointNet[3] + BM | **3.8** | **2.7** | **2.2** | **7.7** | **5.8** | **2.2** | 3700% |
| PointNet++[4] Reg | 3 | 2 | 1.4 | 7.8 | 5.4 | 4.2 | - |
| PointNet++[4] RBF | 3.9 | 2 | 1.5 | 11.2 | 6.2 | 4.8 | 30% |
| PointNet++[4] + BM | **5.7** | **3.7** | **3** | **12.3** | **8.6** | **7** | 85% |
| S3D2 Reg | 5.6 | 3.8 | 3.2 | 10 | 7.7 | 6.8 | - |
| S3D2 RBF[31] | 7.3 | 4.6 | 3.5 | 13 | 8.8 | 7.1 | 30% |
| S3D2 + BM | **11** | **7.5** | **6** | **20** | **13** | **11** | 97% |

**Table 3:** Comparison of Bin-Mixing layers versus regressing parameters directly on Kitti validation set.
Relative values are calculated as % improvement over direct regression in on 3D Moderate setting.

input. Similar to Bin-Mixing layers RBF is applied in per-parameter setting.

### 4.8.3 3D Detection

| Method | Publication | Car 3D mAP | | | Car 2D mAP | | |
|---|---|---|---|---|---|---|---|
| | | Easy | Mod | Hard | Easy | Mod | Hard |
| ROI-10D[18] | CVPR 2019 | 4.32 | 2.02 | 1.46 | 76.56 | 70.16 | 61.15 |
| 3D-GCK[20] | arXiv 2020 | 3.27 | 2.52 | 2.11 | 89.55 | 80.19 | 68.08 |
| GS3D[19] | CVPR 2019 | 4.47 | 2.90 | 2.47 | 86.23 | 76.35 | 62.67 |
| TopNet(LiDAR)[32] | IEEE 2019 | 3.24 | 3.02 | 2.26 | 7.24 | 6.24 | 5.42 |
| RefinedMPL[25] | arXiv 2019 | **18.09** | 11.14 | 8.94 | 88.29 | 65.24 | 53.20 |
| AM3D[24] | ICCV 2019 | 16.50 | 10.74 | **9.52** | **92.55** | **88.71** | **77.78** |
| D4LCN[22] | CVPR 2020 | 16.65 | **11.72** | 9.51 | 90.34 | 83.67 | 65.33 |
| SparVox3D(Ours) | | 5.27 | 3.20 | 2.56 | 83.76 | 67.88 | 52.56 |

**Table 4:** Comparative results of 3D and 2D detection scores on Kitti test set

Comparison of our overall pipeline, SparVox3D to other monocular approaches can be found on 4.8.3. Results are pulled from online leaderboard whose labels are not shared.

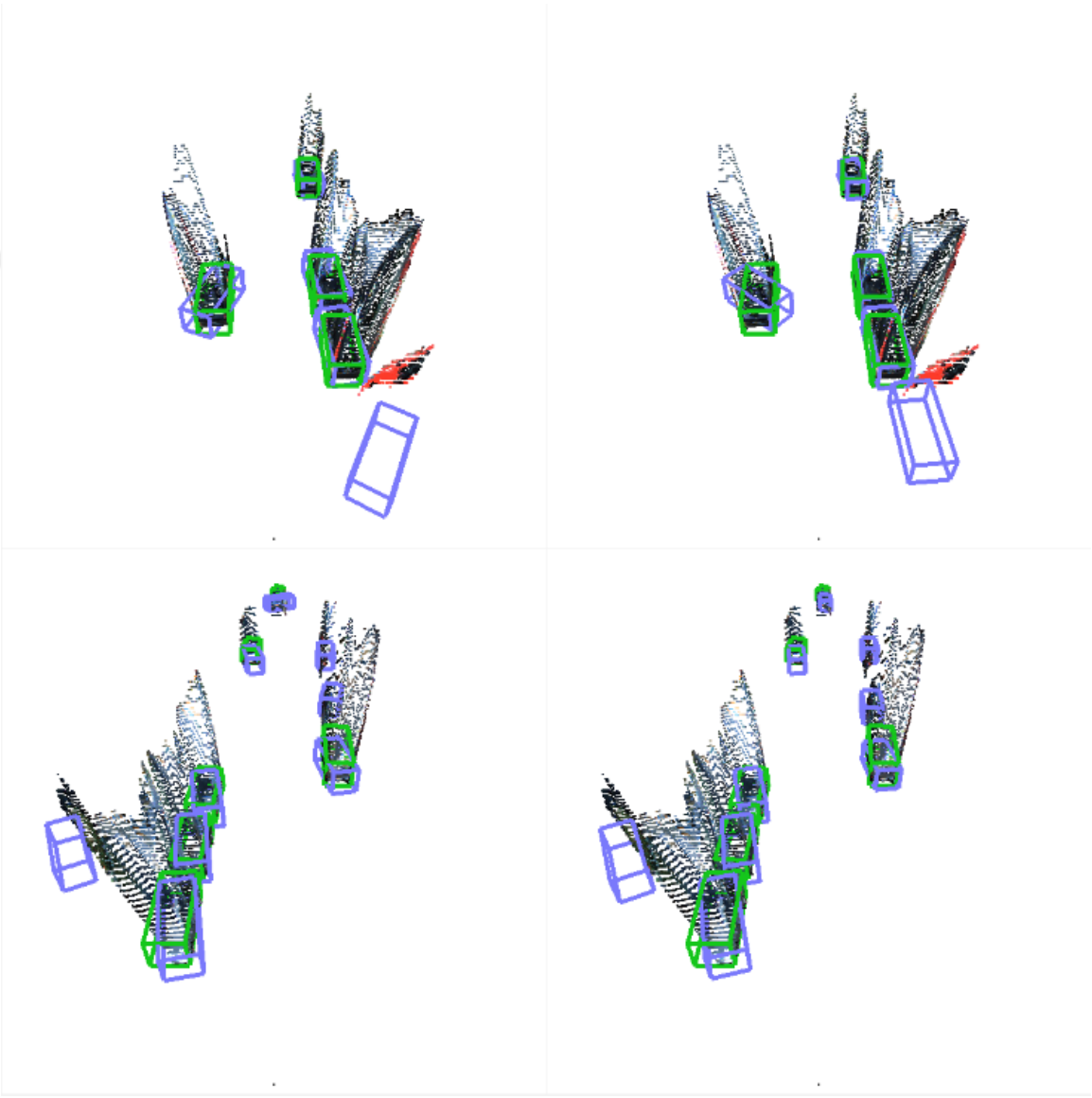

**Figure 18:** Rendering of 3D object detections of SparVox with S3D2 module. Blue color represents predictions, green color represents ground-truth. Left: Bin-Mixing layer enabled, Right: Bin-Mixing layer disabled.

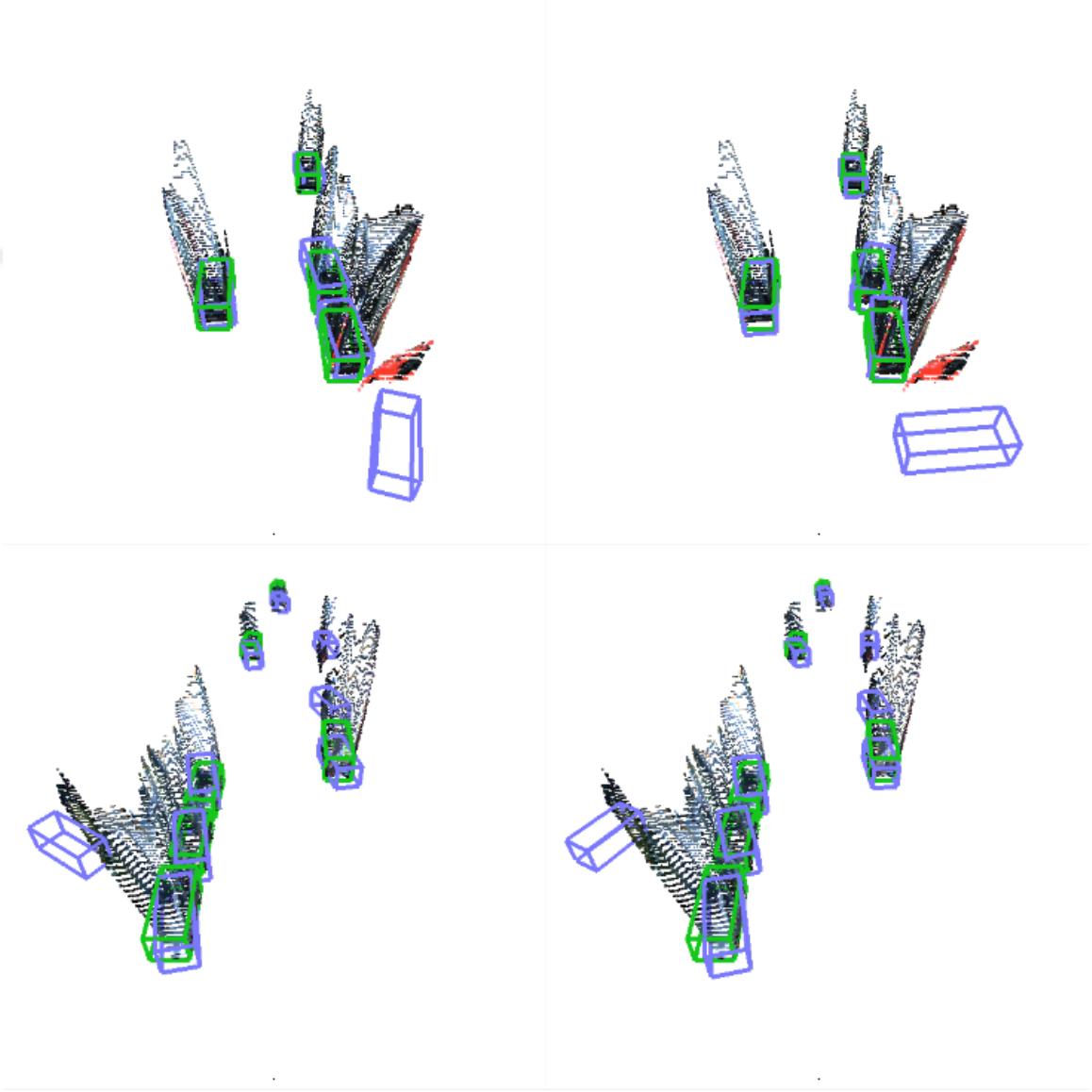

**Figure 19:** Rendering of 3D object detections of SparVox with PointNet++. Blue color represents predictions, green color represents ground-truth. Left: Bin-Mixing layer enabled, Right: Bin-Mixing layer disabled.

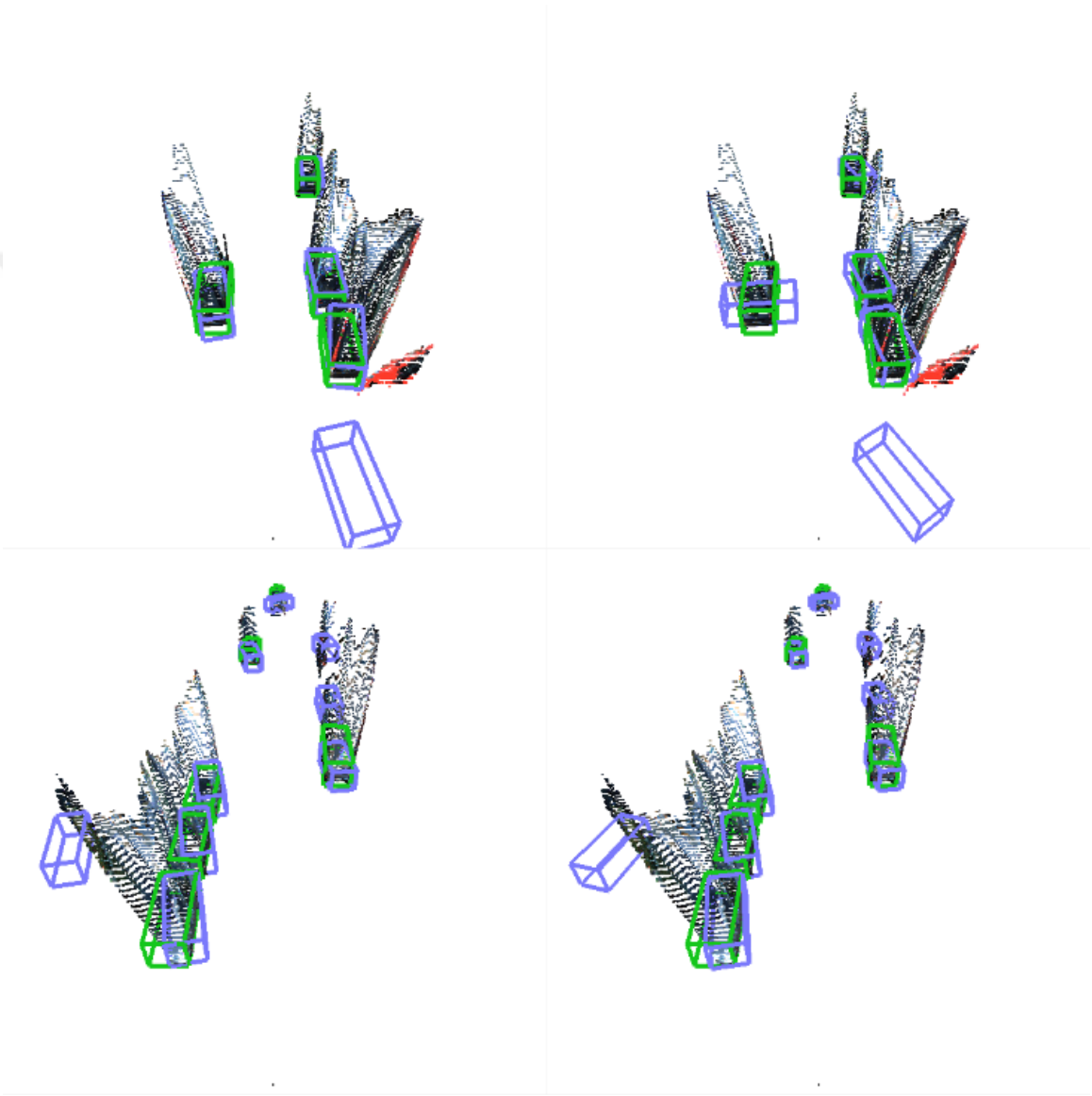

**Figure 20:** Rendering of 3D object detections of SparVox with PointNet. Blue color represents predictions, green color represents ground-truth. Left: Bin-Mixing layer enabled, Right: Bin-Mixing layer disabled.

# CHAPTER V

# CONCLUSION

In this thesis, we tackle the challenging problem of 3D object detection from noisy RGB-D data. We argue that PointNet based 3D information extraction modules cannot effectively extract local patterns and propose to convert data representation into sparse voxel grid and utilize sub-manifold sparse convolutional network instead.

We demonstrate clear advantages of utilizing sparse voxel grid representations combined with sub-manifold sparse convolutions in table 4.8.1. High capacity of convolutional neural networks to analyze spatial local patterns in data gives our proposed S3D2 module a significant lead over PointNet and PointNet++. Although other approaches utilizing PointNet based architectures outperform ours in Kitti leaderboard, we argue that their results are due to better engineered overall pipelines and not because they use PointNet based information extraction modules. By comparing Sub-manifold sparse convolutional neural network architecture to PointNet and PointNet++ directly under the same pipeline, we give a clearer comparison of both methods.

We introduce novel Bin-Mixing layer that transforms the task of regression to bin-weighing problem. Our Bin-Mixing layer makes learning of regression based tasks significantly simpler. This representation of regression problem results in neural network converging at a much better minima and improves final average precision metric by up to 97%. In table 4.8.2 test the effectiveness of Bin-Mixing layer versus regressing parameters directly. Our findings show that in every setting tested, Bin-Mixing layer results in net improvement, we also find that more advanced the design on neural network, higher the benefits of utilizing Bin-Mixing layers are, which gives us clues that

Bin-Mixing layers, by representing the problem in a more natural fashion, can effectively reduce overfitting. More-over, due to simplicity of the design of our Bin-Mixing layer, it can easily be incorporated into any neural network and be implemented in any deep learning framework utilizing only the basic functions.

An argument as into why this different form of problem representation results in a significant improvement over direct regression can be formed as follows; direct regression makes the search in global scope where outputs are unbounded. Bin-Mixing layers localize the output space, where bins are estimates for the highly possible values of a parameter. This localization not only decreases search space greatly, but it also results in a different formulation of the loss where distance between a possible local minima and the global minima is reduced, resulting in network parameters converging on a better local minima. We also demonstrate a lead over RBF based variant of our approach as an alternative source of locality. However, RBF and Bin-Mixing layers are not alternatives to each other. Our findings show on table 4.8.2 that PointNet, without any source of locality, does not even learn a proper mapping between inputs and outputs.

We show that our S3D2 architecture and Bin-Mixing layers can alleviate even a relatively straightforward architecture into getting competitive results in table 4.8.3. Due to our reliance on pre-trained instance segmentation, our 2D detection results fall short of other entries yet our 3D detection results still end up being higher than many, surpassing even a LiDAR based approach.

Compared to other approaches, the main drawback of our approach is its reliance on high quality instance segmentations and depth estimations. Errors made in either of these two stages propagate into our model and without an effective system that can take advantage of error-free image level information, our approach fails to recover from these errors. Our research is preliminary in that it demonstrates advantage of utilizing sparse voxel grid representation and Bin-Mixing layer. We believe significant

improvements are possible through combination of sparse voxel grids, Bin-Mixing layer and image-level information together.

# APPENDIX A

# MODEL ARCHITECTURE

```
Detector(
  (voxel_net): SparseVoxelFeatureExtractor(
    (sparse_voxel_net): SparseSequential(
      (0): SparseSequential(
        (0): SubMConv3d()
        (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (2): LeakyReLU(negative_slope=0.01)
      )
      (1): SparseSequential(
        (0): SparseConv3d()
        (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (2): LeakyReLU(negative_slope=0.01)
      )
      (2): SparseSequential(
        (0): SubMConv3d()
        (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (2): LeakyReLU(negative_slope=0.01)
      )
      (3): SparseSequential(
        (0): SparseConv3d()
        (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (2): LeakyReLU(negative_slope=0.01)
      )
      (4): SparseSequential(
        (0): SubMConv3d()
        (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (2): LeakyReLU(negative_slope=0.01)
      )
      (5): SparseSequential(
        (0): SparseConv3d()
        (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (2): LeakyReLU(negative_slope=0.01)
      )
      (6): SparseSequential(
        (0): SubMConv3d()
        (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (2): LeakyReLU(negative_slope=0.01)
      )
      (7): SparseSequential(
```

```
        (0): SparseConv3d()
        (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (2): LeakyReLU(negative_slope=0.01)
      )
      (8): ToDense()
    )
    (dense_voxel_net): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
      (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
      (5): LeakyReLU(negative_slope=0.01)
      (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
      (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
      (8): LeakyReLU(negative_slope=0.01)
      (9): AdaptiveAvgPool2d(output_size=1)
    )
  )
  (feature_extractor): Sequential(
    (0): Linear(in_features=133, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
    (2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (3): Dropout(p=0.2, inplace=False)
    (4): Linear(in_features=128, out_features=53, bias=True)
  )
  (bin_mixer): BinMixingLayer(
    (softmax): Softmax(dim=1)
  )
)
```

# APPENDIX B

# MODEL CODE

```
import torch
from torch import nn
import numpy as np
import spconv
import configs
import pytorch3d.transforms as p3dt
from utils import cityscapes_utils
import torchsparse.nn as spnn
from torchsparse import SparseTensor
import math

class SparseVoxelFeatureExtractor(nn.Module):
    def __init__(self):
        super(SparseVoxelFeatureExtractor, self).__init__()

        blocks = []
        blocks += self.build_sparse_conv_module(6, 32)   # 32x12x32
        blocks += self.build_sparse_conv_module(32, 64)  # 16x6x6
        blocks += self.build_sparse_conv_module(64, 64)  # 8x3x6
        blocks += self.build_sparse_conv_module(64, 64, 1, 0)  # 8x1x8

        blocks += [
            spconv.ToDense()
        ]

        self.sparse_voxel_net = spconv.SparseSequential(*blocks)

        self.dense_voxel_net = nn.Sequential(
            nn.Conv2d(64, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(),

            nn.Conv2d(64, 64, 3, 1, 0),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(),

            nn.Conv2d(64, 128, 3, 1, 0),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(),

            nn.AdaptiveAvgPool2d(1)
        )

    def build_sparse_conv_module(self, in_f, out_f, max_pool_kernel_size=2, padding=1):
        module = [spconv.SparseSequential(
            spconv.SubMConv3d(in_f, out_f, 3, 1, 1, bias=False),
```

```
            nn.BatchNorm1d(out_f),
            nn.LeakyReLU()
        )]

        if max_pool_kernel_size != 1 or padding != 1:
            module.append(
                spconv.SparseSequential(
                    spconv.SparseConv3d(out_f, out_f, 3, max_pool_kernel_size, padding),
                    nn.BatchNorm1d(out_f),
                    nn.LeakyReLU(),
                )
            )

        return module

    def forward(self, locations, features, batch_size):
        spatial_shape = (np.array(configs.voxel_map_size) / configs.voxel_size)
        spatial_shape = [math.ceil(x) for x in spatial_shape]
        x = spconv.SparseConvTensor(features.to(configs.device), locations.cpu(), spatial_shape, batch_size)
        x = self.sparse_voxel_net(x)
        # print(x.shape)
        x = x.squeeze(3)
        # print(x.shape)
        x = self.dense_voxel_net(x)
        # print(x.shape)

        x = x.flatten(1)
        return x


class BinMixingLayer(nn.Module):
    def __init__(self):
        super(BinMixingLayer, self).__init__()
        self.softmax = nn.Softmax(1)

        # Bins are automatically generated in precisely the same format from Jupyter notebook, just copy paste them

        self.bins = {}
        self.bins['x'] = [-4.801, -1.222, -0.04, 0.911, 4.195]
        self.bins['y'] = [-0.676, -0.072, 0.079, 0.343, 1.3]
        self.bins['z'] = [-7.062, -4.326, -2.514, -1.325, -0.459, 0.315, 0.99, 1.905, 3.857, 7.032]
        self.bins['width'] = [0.199, 0.626, 1.543, 1.685, 2.131]
        self.bins['height'] = [1.029, 1.425, 1.566, 1.792, 2.361]
        self.bins['length'] = [-0.304, 0.902, 3.51, 4.222, 7.304]
        self.bins['y_rot'] = [-2.197, -0.159, 1.344, 3.141, 4.46, 5.339]

        for key in self.bins:
            self.bins[key] = torch.Tensor([self.bins[key]]).to(configs.device).transpose(1, 0)

    def forward(self, x):
        x_votes = self.softmax(x[:, 0:5])
        y_votes = self.softmax(x[:, 5:10])
        z_votes = self.softmax(x[:, 10:20])
        width_votes = self.softmax(x[:, 20:25])
        height_votes = self.softmax(x[:, 25:30])
```

```
        length_votes = self.softmax(x[:, 30:35])
        y_rot_votes = self.softmax(x[:, 35:41])

        x_loc = torch.matmul(x_votes, self.bins['x'])
        y_loc = torch.matmul(y_votes, self.bins['y'])
        z_loc = torch.matmul(z_votes, self.bins['z'])
        width = torch.matmul(width_votes, self.bins['width'])
        height = torch.matmul(height_votes, self.bins['height'])
        length = torch.matmul(length_votes, self.bins['length'])
        y_rot = torch.matmul(y_rot_votes, self.bins['y_rot'])

        outputs = torch.cat([x_loc, y_loc, z_loc, width, height, length, y_rot], 1)

        return outputs


class Detector(nn.Module):
    def __init__(self):
        super(Detector, self).__init__()

        if configs.DETECTOR_MODE == configs.MODE_SPARSE_VOX:
            self.voxel_net = SparseVoxelFeatureExtractor().float()
        elif configs.DETECTOR_MODE == configs.MODE_POINTNET:
            self.point_net = PointNetCls(k=128)
        else:
            self.point_net2 = PointNet2ClassificationMSG({"model.use_xyz": True}).to(configs.device)

        feature_count = -1
        if configs.DETECTOR_MODE == configs.MODE_POINTNET_PLUS:
            feature_count = 40
        else:
            feature_count = 128

        if configs.BIN_MIXING_LAYER_ENABLED:
            self.feature_extractor = nn.Sequential(
                torch.nn.Linear(feature_count + 3 + len(configs.valid_objects), 128),
                torch.nn.LeakyReLU(),
                torch.nn.BatchNorm1d(128),
                torch.nn.Dropout(0.2),
                torch.nn.Linear(128, 53)
            ).float()
        else:
            self.feature_extractor = nn.Sequential(
                torch.nn.Linear(feature_count + 3 + len(configs.valid_objects), 128),
                torch.nn.LeakyReLU(),
                torch.nn.BatchNorm1d(128),
                torch.nn.Dropout(0.2),
                torch.nn.Linear(128, 7)
            ).float()

        self.bin_mixer = BinMixingLayer()

    def normalize_output_angle(self, angles):
        angles = torch.where(angles > np.pi / 2, np.pi - angles, angles)
        angles = torch.where(angles < -np.pi / 2, -np.pi - angles, angles)
```

```
        return angles

    def forward(self, ip, centers, type_oht_encodings):
        if configs.DETECTOR_MODE == configs.MODE_SPARSE_VOX:
            locations, features = ip["locations"], ip["features"]
            batch_size = ip["batch_size"]
            features = self.voxel_net(locations, features, batch_size)
        elif configs.DETECTOR_MODE == configs.MODE_POINTNET:
            ip = torch.Tensor(ip).to(configs.device)
            ip = ip.permute(0, 2, 1)
            features = self.point_net(ip)
        else:
            # print(ip.shape)
            ip = torch.Tensor(ip).to(configs.device)
            features = self.point_net2(ip)
            # print(ip.shape, features.shape)

        features = torch.cat((features, centers / 80, type_oht_encodings), 1)
        features = self.feature_extractor(features)

        if configs.BIN_MIXING_LAYER_ENABLED:
            predictions = self.bin_mixer(features)
        else:
            predictions = features * configs.LOCAL_DISTANCE_NORMALIZATION_FACTOR

        predictions[:, :3] = predictions[:, :3] + centers

        return predictions


class DetectorWrapper:
    def __init__(self):

        self.detector = Detector().to(configs.device)

        self.lr = 0.001
        self.lr_gamma = 0.1
        self.lr_steps = 180

        self.optimizer_algo = torch.optim.Adam
        self.optimizer = self.optimizer_algo(self.detector.parameters(), lr=self.lr)
        # self.scheduler = torch.optim.lr_scheduler.StepLR(self.optimizer, self.lr_steps, self.lr_gamma)

        self.loss = torch.nn.SmoothL1Loss(reduction="none")

        self.epoch = 0
        self.epoch_step = 0

        self.global_train_step = 0

    def eval(self):
        self.detector.eval()

    def train(self):
        self.detector.train()
```

```
def get_state_dict(self):
    return {
        "optimizer": self.optimizer.state_dict(),
        # "scheduler": self.scheduler.state_dict(),
        "detector": self.detector.state_dict(),
        "epoch": self.epoch,
        "epoch_step": self.epoch_step,
        "global_train_step": self.global_train_step
    }

def load_state_dict(self, dict):
    self.epoch = dict["epoch"]
    self.epoch_step = dict["epoch_step"]
    self.global_train_step = dict["global_train_step"]

    self.detector.load_state_dict(dict["detector"])
    self.optimizer = self.optimizer_algo(self.detector.parameters(), self.lr)
    self.optimizer.load_state_dict(dict["optimizer"])

    # self.scheduler = torch.optim.lr_scheduler.StepLR(self.optimizer, self.lr_steps, self.lr_gamma,
    #                                                  last_epoch=self.epoch)
    # self.scheduler.load_state_dict(dict["scheduler"])

def save(self, filepath):
    torch.save(self.get_state_dict(), filepath)

def load(self, filepath):
    checkpoint = torch.load(filepath)
    self.load_state_dict(checkpoint)

def calculate_loss(self, pred, label):
    loss = torch.sum(torch.mean(self.loss(pred, label), 1))

    if configs.CORNER_LOSS_ENABLE_EPOCH != -1 and self.epoch >= configs.CORNER_LOSS_ENABLE_EPOCH:
        corner_loss = self.calculate_corner_loss(pred, label) / 10.0
        loss += corner_loss

    return loss

def get_corner_points(self, bbox3d, angle_offset=0):
    corner_points = bbox3d[:, 3:6].unsqueeze(1).repeat(1, 8, 1) / 2
    corner_centers = bbox3d[:, :3].unsqueeze(1).repeat(1, 8, 1)
    corner_points[:, 1, 0] *= -1

    corner_points[:, 2, 1] *= -1

    corner_points[:, 3, 2] *= -1

    corner_points[:, 4, 0:2] *= -1

    corner_points[:, 5, 1:3] *= -1

    corner_points[:, 6, 0] *= -1
    corner_points[:, 6, 2] *= -1
```

```
        corner_points[:, 7, :] *= -1

        rotation = p3dt.RotateAxisAngle(bbox3d[:, -1] + angle_offset, axis='Y').cuda()
        corner_points = rotation.transform_points(corner_points) + corner_centers

        corner_points += corner_centers

        return corner_points

    def get_point_distance(self, pred, target):
        distances = torch.sum((pred - target)**2, 2)**0.5
        distances = torch.sum(distances, 1)
        return distances

    def calculate_corner_loss(self, pred, label):
        corner_points1 = self.get_corner_points(pred)
        corner_points2 = self.get_corner_points(pred, np.pi)
        target_points = self.get_corner_points(label)

        losses1 = self.get_point_distance(corner_points1, target_points).reshape(-1, 1)
        losses2 = self.get_point_distance(corner_points2, target_points).reshape(-1, 1)

        losses = torch.cat([losses1, losses2], 1)
        min_losses, _ = torch.min(losses, 1)

        return torch.mean(min_losses)

    def train_step(self, ip, centers, type_oht_encodings, label):
        self.global_train_step += 1
        pred = self.detector(ip, centers, type_oht_encodings)
        loss = self.calculate_loss(pred, label)

        loss.backward()
        self.optimizer.step()
        self.optimizer.zero_grad()

        return loss.item(), pred

    def predict(self, ip, center, type_oht_encodings):
        pred = self.detector(ip, center, type_oht_encodings)
        return pred
```

# APPENDIX C

# POINT CLOUD TO SPARSE VOXEL REPRESENTATION CODE

```
def generate_sparse_voxel_representation(points, center):
    v_size = configs.voxel_size
    v_area = configs.voxel_map_size

    voxel_map_size = np.asarray(np.ceil(np.array(v_area) / v_size), np.int32)
    center_x, center_y, center_z = center

    x_begin, x_end = [center_x + sign * 0.5 * voxel_map_size[0] * v_size for sign in [-1, 1]]
    y_begin, y_end = [center_y + sign * 0.5 * voxel_map_size[1] * v_size for sign in [-1, 1]]
    z_begin, z_end = [center_z + sign * 0.5 * voxel_map_size[2] * v_size for sign in [-1, 1]]

    sparse_dict = {}
    for point in points:
        x, y, z, r, g, b = point

        if x_begin < x < x_end and y_begin < y < y_end and z_begin < z < z_end:
            z_coord = math.floor((x - x_begin) / v_size)
            y_coord = math.floor((y - y_begin) / v_size)
            x_coord = math.floor((z - z_begin) / v_size)

            key = (0, z_coord, y_coord, x_coord)
            if key in sparse_dict.keys():
                sparse_dict[key] += [r, g, b, x, y, z, 1]
            else:
                sparse_dict[key] = np.array([r, g, b, x, y, z, 1])

    for key in sparse_dict.keys():
        sparse_dict[key] = sparse_dict[key][:-1] / sparse_dict[key][-1]
        sparse_dict[key][3:6] -= center
        sparse_dict[key][3:6] /= v_area

    return {"locations": np.array(list(sparse_dict.keys())), "features": np.array(list(sparse_dict.values())),
            "batch_size": 1}
```

# APPENDIX D

# RGB-D TO PSEUDO-LIDAR CODE

```
def convert_depthmap_to_points(depth_map, focal_length_in_pixels=None,
                               principal_point=None,
                               rgb_image=None,
                              is_depth_along_z=True):
    '''
    Converts given depth map to point cloud
    :param depth_map: A grayscale image where values are equal to depth in meters
    :param focal_length_in_pixels: Focal lenght of the camera measured in pixels
    :param principal_point: Center of image
    :param rgb_image: Colored image that matches depth map, used for coloring points
    :param is_depth_along_z: If True, z coordinate of the pixel will be equal to depth,
    otherwise depth will be calculated as distance between camera and pixel
    :return: List of points where each point is of form [x, y, z, r, g, b]
    '''
    if focal_length_in_pixels is None:
        focal_length_in_pixels = 715
    if principal_point is None:
        principal_point = [depth_map.shape[0] / 2, depth_map.shape[1] / 2]

    if depth_map.shape[0] == 1:
        depth_map = np.swapaxes(depth_map, 0, 1).swapaxes(1, 2)

    points = np.ones(shape=(depth_map.shape[0] * depth_map.shape[1], 6))  # A point contains x,y,z,r,g,b
    if rgb_image is None:
        points[:, 3:6] = [0.5, 0.7, 1]
    else:
        if rgb_image.shape[0] == 3:
            rgb_image = np.swapaxes(rgb_image, 0, 1).swapaxes(1, 2)
        rgb_image = rgb_image.reshape(-1, 3)
        points[:, 3:6] = rgb_image / 256.0

    y, x = np.meshgrid(np.arange(0, depth_map.shape[1]), np.arange(0, depth_map.shape[0]))
    yx_coordinates = np.array([x.flatten(), y.flatten()], np.float32).T
    yx_coordinates += -1 * np.array(principal_point)
    yx_coordinates = np.flip(yx_coordinates, 1)

    points[:, 0:2] = yx_coordinates
    points[:, 2] = depth_map.flatten()

    pixel_dist = (points[:, 0] ** 2 + points[:, 1] ** 2) ** 0.5
    focal_target_dist = (focal_length_in_pixels ** 2 + pixel_dist ** 2) ** 0.5

    if not is_depth_along_z:
        points[:, 2] = points[:, 2] * focal_length_in_pixels / focal_target_dist
```

```
    points[:, 0] = points[:, 0] * points[:, 2] / focal_length_in_pixels
    points[:, 1] = points[:, 1] * points[:, 2] / focal_length_in_pixels

    points[:, 1] *= -1
    return points
```

# Bibliography


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

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

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

[4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, pp. 5099–5108, 2017.

[5] B. Graham and L. van der Maaten, "Submanifold sparse convolutional networks," *arXiv preprint arXiv:1706.01307*, 2017.

[6] K. Hornik, M. Stinchcombe, H. White, *et al.*, "Multilayer feedforward networks are universal approximators.," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[7] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[8] S. Jadon, "Introduction to different activation functions for deep learning," *Medium, Augmenting Humanity*, vol. 16, 2018.

[9] A. LeNail, "Nn-svg: Publication-ready neural network architecture schematics," *Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

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

[12] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[13] O. Alexandrov, Nov 2004.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[15] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural Networks for Machine Learning, 2012.

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

[17] L. Liu, J. Lu, C. Xu, Q. Tian, and J. Zhou, "Deep fitting degree scoring network for monocular 3d object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1057–1066, 2019.

[18] F. Manhardt, W. Kehl, and A. Gaidon, "Roi-10d: Monocular lifting of 2d detection to 6d pose and metric shape," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2069–2078, 2019.

[19] B. Li, W. Ouyang, L. Sheng, X. Zeng, and X. Wang, "Gs3d: An efficient 3d object detection framework for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1019–1028, 2019.

[20] N. Gählert, J.-J. Wan, N. Jourdan, J. Finkbeiner, U. Franke, and J. Denzler, "Single-shot 3d detection of vehicles from monocular rgb images via geometry constrained keypoints in real-time," *arXiv preprint arXiv:2006.13084*, 2020.

[21] P. Li, H. Zhao, P. Liu, and F. Cao, "Rtm3d: Real-time monocular 3d detection from object keypoints for autonomous driving," *arXiv preprint arXiv:2001.03343*, 2020.

[22] M. Ding, Y. Huo, H. Yi, Z. Wang, J. Shi, Z. Lu, and P. Luo, "Learning depth-guided convolutions for monocular 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1000–1001, 2020.

[23] X. Weng and K. Kitani, "Monocular 3d object detection with pseudo-lidar point cloud," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[24] X. Ma, Z. Wang, H. Li, P. Zhang, W. Ouyang, and X. Fan, "Accurate monocular 3d object detection via color-embedded 3d reconstruction for autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 6851–6860, 2019.

[25] J. M. U. Vianney, S. Aich, and B. Liu, "Refinedmpl: Refined monocular pseudolidar for 3d object detection in autonomous driving," *arXiv preprint arXiv:1911.09712*, 2019.

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

[27] H. Chen, K. Sun, Z. Tian, C. Shen, Y. Huang, and Y. Yan, "Blendmask: Top-down meets bottom-up for instance segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8573–8581, 2020.

[28] Z. Tian, H. Chen, X. Wang, Y. Liu, and C. Shen, "AdelaiDet: A toolbox for instance-level recognition tasks." https://git.io//adelaidet, 2019.

[29] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 806–814, 2015.

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

[31] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels, "On the training of radial basis function classifiers," *Neural networks*, vol. 5, no. 4, pp. 595–603, 1992.

[32] S. Wirges, M. Reith-Braun, M. Lauer, and C. Stiller, "Capturing object detection uncertainty in multi-layer grid maps," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1520–1526, IEEE, 2019.

[33] D. E. Knuth, *The TEX Book*. Reading, Massachusetts: Addison-Wesley, 1984. Reprinted as Vol. A of *Computers & Typesetting*, 1986.

[34] D. E. Knuth, *TEX: The Program*, vol. B of *Computers & Typesetting*. Reading, Massachusetts: Addison-Wesley, 1986.

[35] D. E. Knuth, "The WEB system for structured documentation, version 2.3," Tech. Rep. STAN-CS-83-980, Computer Science Department, Stanford University, Stanford, California, Sept. 1983.

[36] D. E. Knuth, "Literate programming," *The Computer Journal*, vol. 27, pp. 97–111, May 1984.

[37] D. E. Knuth, "A torture test for TEX, version 1.3," Tech. Rep. STAN-CS-84-1027, Computer Science Department, Stanford University, Stanford, California, Nov. 1984.

[38] R. K. Furuta and P. A. MacKay, "Two TEX implementations for the IBM PC," *Dr. Dobb's Journal*, vol. 10, pp. 80–91, Sept. 1985.

[39] J. Désarménien, "How to run TEX in french," Tech. Rep. SATN-CS-1013, Computer Science Department, Stanford University, Stanford, California, Aug. 1984.

[40] A. L. Samuel, "First grade TEX: A beginner's TEX manual," Tech. Rep. SATN-CS-83-985, Computer Science Department, Stanford University, Stanford, California, Nov. 1983.

[41] L. Lamport, *LATEX: A Document Preparation System. User's Guide and Reference Manual*. Reading, Massachusetts: Addison-Wesley, 1986.

[42] M. D. Spivak, *The Joy of TEX*. American Mathematical Society, 1985.

[43] O. Patashnik, *BibTEXing*. Computer Science Department, Stanford University, Stanford, California, Jan. 1988. Available in the BibTEX release.

[44] O. Patashnik, *Designing BibTEX Styles*. Computer Science Department, Stanford University, Jan. 1988.

[45] D. Fuchs, "The format of TEX's DVI files version 1," *TUGboat*, vol. 2, pp. 12–16, July 1981.

[46] D. Fuchs, "Device independent file format," *TUGboat*, vol. 3, pp. 14–19, Oct. 1982.

[47] F. Manhardt, W. Kehl, and A. Gaidon, "Roi-10d: Monocular lifting of 2d detection to 6d pose and metric shape," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2069–2078, 2019.

[48] L. Liu, J. Lu, C. Xu, Q. Tian, and J. Zhou, "Deep fitting degree scoring network for monocular 3d object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1057–1066, 2019.

# VITA

Eren Balatkan received his Bachelors degree from Özyeğin University at 2018. After receiving his Bachelors, he started pursuing his M.Sc. degree at Özyeğin University. Under supervision of Assist. Prof. Dr. M. Furkan Kıraç, he focused his research on deep learning based computer vision. Eren's research interests include artificial intelligence, computer graphics and simulations.