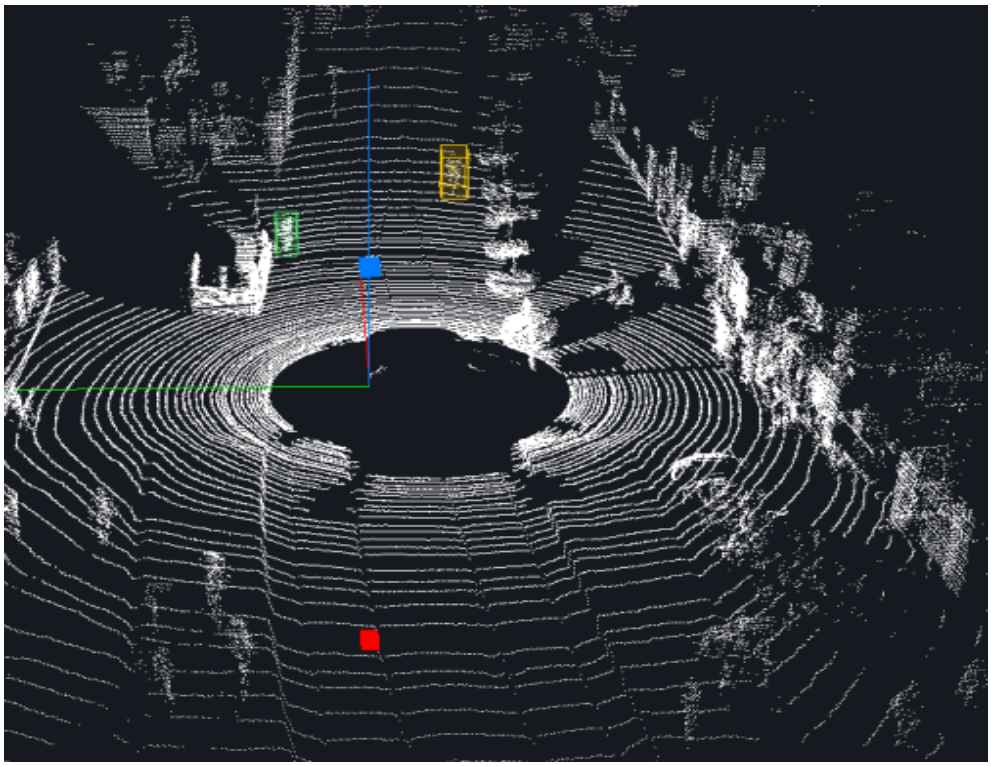
Robotics Research Lab
Department of Computer Science
University of Kaiserslautern

Master Thesis



3D Object Detection With LiDAR In Deep Learning

Surendra Kumar Aralapura Mariyappa

December 26, 2021

Master Thesis

3D Object Detection With LiDAR In Deep Learning

Robotics Research Lab
Department of Computer Science
University of Kaiserslautern

Surendra Kumar Aralapura Mariyappa

Day of issue : 15.11.2020
Day of release : 05.08.2021

Reviewer : Prof. Dr. Karsten Berns
Supervisor : M.Sc. Axel Vierling

Hereby I declare that I have self-dependently composed the Master Thesis at hand. The sources and additives used have been marked in the text and are exhaustively given in the bibliography.

December 26, 2021 – Kaiserslautern

(Surendra Kumar Aralapura Mariyappa)

Acknowledgement

I would like to express my gratitude to Prof. Dr. Karsten Berns, and my mentor M.Sc. Axel Vierling for providing this opportunity to write my master thesis at RR Lab.

I am extremely thankful to my mentor for supporting me through lots of discussions and suggestions. Even with his busy schedules, Axel Vierling has helped me to be motivated and to understand the core concepts and to consider all the possible scenarios.

I would like to thank my mentor once again and RR Lab for all the support and hardware.

I also want to thank my friends, especially M.Sc. Asmita Mittal, Jakub Pawlak, and my roommate Ajay Chawda for the healthy discussion during my thesis and finally, my family for their support.

Abstract

Object detection is a well-studied and researched topic in computer vision and deep learning(DL). The application of object detection can be found in different domains such as autonomous driving, surveillance, machine inspection, housekeeping robots and so on. This work explores the problem with 3D object detection with a stereo or multi-view camera to use in the real-time autonomous driving application and the way of fixing them through new sensing modalities like LiDAR. In this work, the base network of the State-of-the-Art(SOTA) models in 3D object detection with LiDAR is explored, as a supplement to a camera-based 3D object detection system. The network is evaluated for different situations such as the addition of noise in the input data, single-class versus multi-label, various distances 32 meters (m), and 48 m. Additionally, examination of the existing LiDAR ground-truth annotation tools was conducted along with providing an approach to solve the class imbalance problem in 3D object detection with LiDAR. Also, a rule of thumb on choosing one of the important hyper-parameters in the model is proposed. In the end, results of the above-mentioned experiments on the model are presented along with providing a glimpse of recent advances in 3D object detection with LiDAR.

Contents

1. Introduction

Autonomous Driving(AD) is a way of achieving the vehicle to drive us, instead of being driven by a human. The potential benefits of AD would be saving people's lives, where most of the driving accidents are due to human error[USDT 21]. Additionally, we can spend time more productively rather than driving.

Any AD either car or truck, relies on four fundamental concepts: environmental perception and modeling, localization and mapping, path-planning and decision making, and motion control, as shown in Figure 1.1. The term environmental perception would be the first to be considered to provide the next core functionalities to AD. The way of interpreting its surrounding by vehicle is known as environmental perception. Firstly, the interpretation should be accurate and reliable, which means both the interpreted environment and actual environment are the same. Secondly, the interpretation should be fast to use in real-time. If not, the processed information is irrelevant.

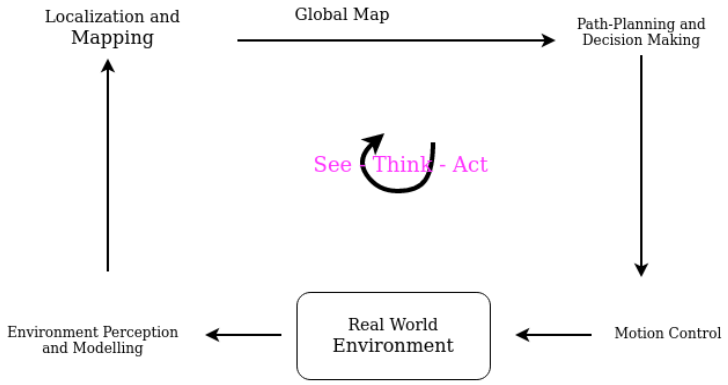


Figure 1.1: The fundamental concepts in autonomous driving.

There has been lots of research and development work going on in developing fully autonomous vehicles, which is level 5. Level 5 vehicles will not have a steering wheel, acceleration pedal, braking pedals, and driver's seat in the vehicle along with no human intervention while driving in a dynamic environment. But the fundamental problem is,

which and how many sensors to use for environmental perception? Does the camera alone solve the perception problem or does an autonomous vehicle(AV) need a LiDAR along with the camera?

This work will focus on the sub-problem of environmental perception, i.e., object detection, along with addressing the issues with only camera based 3D object detection system in AV. Object detection is the way of classifying and localizing the object in a given environment. While driving, the vehicle should react to the dynamic objects(moving from point A to point B), which are safety-critical like pedestrians and cyclists. The objective is to detect and localize the objects, which differ in size and appearance in complex and varying driving conditions. One of the promising technology, which can address the object detection is deep learning(DL). Basis of this will be discussed in section 2.2.

There has been significant progress in 2D object detection using DL. In 2D object detection, input is image of size, NxMx3 or NxMx1, in case of RGB image or gray image respectively, and the output is an object class along with the position of the object in an image($x_{TopLeft}$, $y_{TopLeft}$, $x_{BottomRight}$, $y_{BottomRight}$). The State-Of-The-Art(SOTA) models in 2D object detection will be discussed in section 3.1. But for the proper control of AD, an AV need the object detection in 3D as well, where the distance to the objects is available. The input to the 3D object detection system is point cloud either from light detection and ranging(LiDAR) or 3D vision sensors. Although, distance to the objects is very limited in 3D vision sensors.

Currently, there exist lots of deep learning-based models for 3D object detection with LiDAR. In 3.2, we will discuss the current SOTA models including different 3D feature extractors. When working with supervised-based models in deep learning, getting the annotated data, i.e. ground truth data, is important step. Unlike annotation of image data, annotation of 3D data is more complex and time-consuming. The requirements for the 3D data annotation will be presented in section 3.3. The chosen model in this thesis along with changes made to the model will be explained in section 4. In section 5, the possible solution to tackle the class imbalance problem in 3D object detection will be discussed. Finally, experimentation on the model and followed by results and analysis of experimentation will be presented in section 6.

1.1 Problem Formulation

The object detection in 3D in an autonomous vehicle perception stack serves as a basis for path planning, motion control, and avoidance of safety-critical objects. Normally, stereo or multi-view camera systems are used for 3D object detection in AD. But the problem with camera-based 3D object detection is; complex matching algorithm, high computational time, low spatial resolution, vulnerability to illumination and weather conditions, limited field of view(FOV), limited range information, reliability of distance information goes down as the distance increases(more than 30 meters) [Kumar 20].

In recent years, 3D object detection with LiDAR got the attention of researchers to provide a supplementary 3D detection system to an AV, which partially complements the perception stack of AV. But, most of the deep learning-based 3D object detection models [Ku 18], and [Chen 17] use both LiDAR and RGB data as input to a detection model. However, the above-mentioned methods add further complication to the perception stack of an AV. The complications are:

- 3D detection system with LiDAR again depends on camera data in case of [Ku 18]
- Calibration and synchronization between camera and LiDAR (voxelnet based late fusion model) [Ku 18]
- Not good for smaller objects like pedestrian and cyclist [Chen 17]
- Loss of object information [Simony 18]

Therefore, to provide a robust, independent, and safe, 3D object detection system to an AV, only LiDAR data as input should be considered. Let's consider a scenario, an AV is driving on its own, if the 3D object detection system depends on camera data in detecting the long-distance(more than 30 meter) and safety-critical objects like pedestrian and cyclist, to take necessary control action, how well the detection system performs during the dark condition, heavy lighting or calibration miss-match between camera and LiDAR? This work focuses mainly on providing LiDAR only based 3D object detection system, where point cloud as input and pedestrians and cyclists as the main objects to be detected.

1.2 Goal of Thesis

The thesis aims at presenting the architecture of the 3D object detection model using DL along with the main challenges in using only LiDAR data as an input, which is highly sparse and unordered.

The following are the goal of this thesis:

- Preprocessing the LiDAR point cloud data.
- Model evaluation on different data-sets from two different domains, when considering both pedestrian and cyclist at the same time as compared to only pedestrian
- What are the difficulties in LiDAR data annotations?
- How can we tackle the class imbalance problem in 3D object detection?
- How can we use the 3D object detection model in real time?

2. Theory

This chapter presents the basic concepts that are the basis of this thesis. Firstly, We will go through the different sensors used in autonomous driving, then a brief introduction of the artificial neural networks followed by different activation functions, optimizers, and convolutional neural networks.

2.1 Sensors

We, as a human, interpret the real world through our visual system. Likewise, a machine must get some kind of information to understand its surroundings. The collection of real-world data is commonly achieved by using different sensors for different types of information.

2.1.1 Camera

Cameras are widely used to gather information from the real world in the form of images. Images are formed by projecting captured 3D real-world scenes onto a 2D plane. The projected locations are saved as RGB pixel values. Therefore, there will be a loss of depth information. Also, cameras are very sensitive to light variations and weather conditions. One of the important tasks in AD is obstacle perception and avoidance, which is indeed possible with stereo cameras. But complex and expensive image matching problem adds an additional computation burden on already existing camera-based perception systems. Moreover, due to the limited range information using stereo or multi-view systems and considering the above-mentioned drawbacks, a camera-based 2D/3D object detection system could not be used alone in AV.

2.1.2 LiDAR

LiDAR stands for "light detection and ranging". It gathers data by emitting pulsed light waves into the surrounding environment, then those pluses rebound back and return to the sensor. Using the time taken by each pulse to bounce back from the objects to the sensor, distance to the objects is calculated. The reflected pulse is stored as a data point,

the collection of data point referred to as a point cloud. Each data point is saved as x, y, z(in cartesian coordinates) in 3D space along with reflectance r. Unlike stereo and mono cameras, LiDAR is not sensitive to light variations, but it is sensitive to weather conditions like rain and snow. Another important difference between the stereo camera and LiDAR is, point cloud from LiDAR will be sparse as compared to the dense nature of the image from a stereo camera.

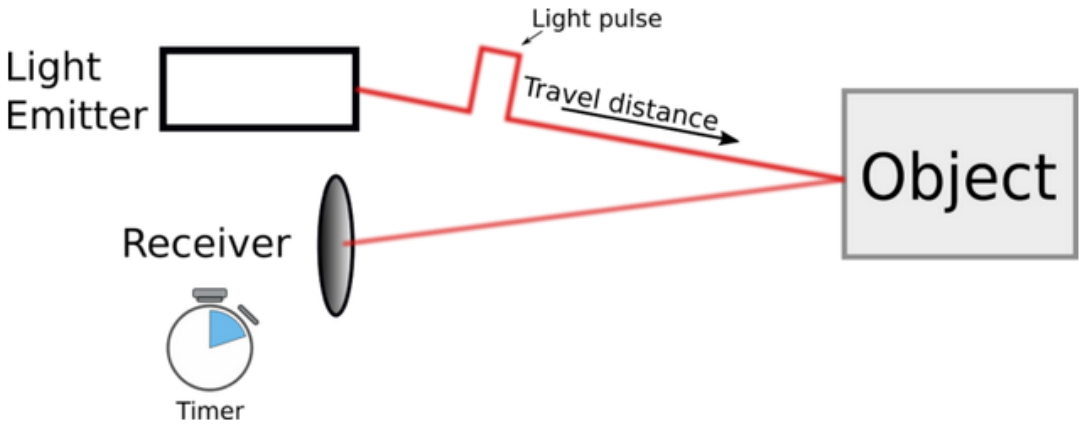


Figure 2.1: Working principle of LiDAR: Pulse emission. The receiver contains an electronic circuit that detects the reflected pulse and calculates its runtime.

2.2 Artificial Neural Networks

The neurons are the basic information-processing unit of the brain. The brain contains billions of neurons that are connected to each other. Each neuron receives an electrical impulse from dendrites and outputs the signal along its axon. Eventually, the axon branches out and connects to the dendrites of other neurons through synapses. The strength of the synapse defines the influence of neurons on other neurons. It is important to note that not all the dendrites will have the same effect on the neuron. Finally, all the signals coming through dendrites into a cell body will be summed. If the final sum is above a certain threshold, then the neuron can send the signal through the axon. This process is also called as activation of the neuron.

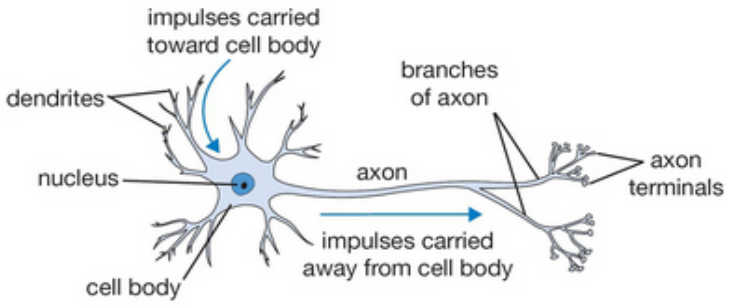


Figure 2.2: Biological representation of neuron[Olof Berg Marklund 21]

.

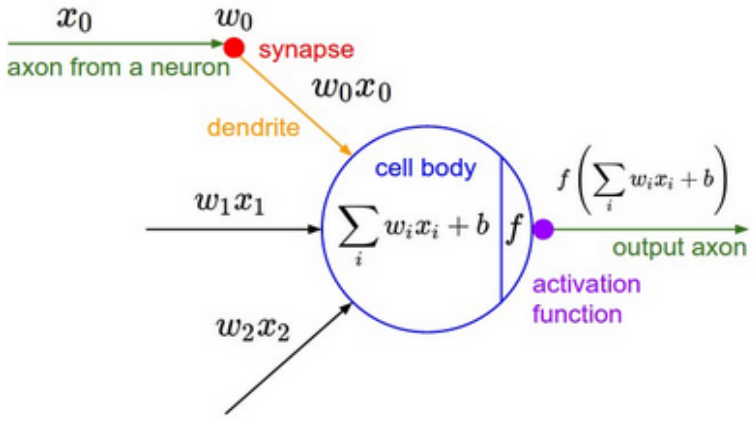


Figure 2.3: Mathematical model of neuron[Olof Berg Marklund 21]

Inspired by the modeling of the human brain system, an artificial neural network(ANN) was designed. ANN also consists of a computational node called 'neuron'. The group of neurons is called multiple perceptrons. The organization of multiple perceptrons in different layers is called multi-layer perceptrons(MLP) as shown in the Figure 2.4. There are three layers mainly, input layer, hidden layer, and output layer.

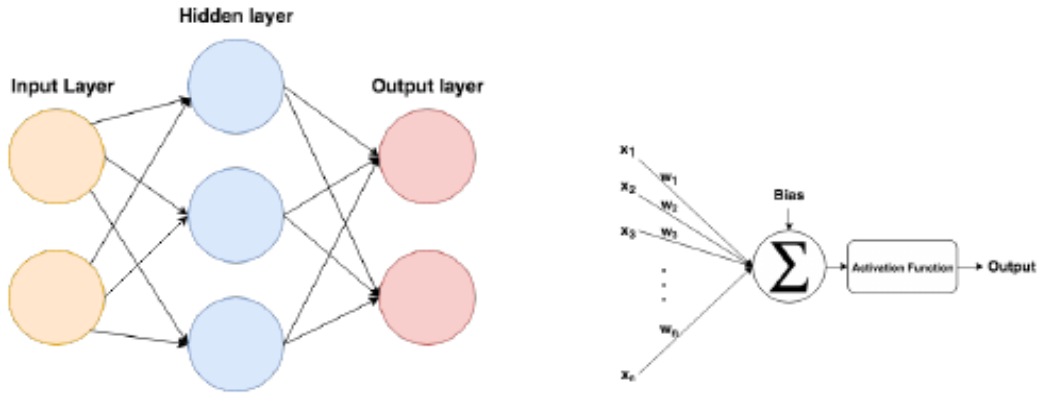


Figure 2.4: A simple example of flow of data from input layer to output layer through hidden layer along with computational representation.

- Input layer: The purpose of input layer is to store and represent the data which will be fed into the layers after it.

- Hidden layer: This layer lies in between input and output layers. Each neuron from the input layer will be connected to each neuron in the hidden layer. The purpose of using a hidden layer is to add non-linearity to the data through activation functions. Unlike the input layer, there will be computation on the data from the input layer/any previous layer.

- Output layer: The input to the output layer neuron is from the previous hidden layer. After doing computation on data, the output layer provides specific data, which will be useful depending upon the task.

In multi-layer perceptrons, each neuron is connected to all the neurons in the next layer through weights. Weights define the influence of neurons on other neurons in the next layer. Then an activation function will be applied to the weighted sum of neurons associated with bias b. Here, the activation function defines the frequency of activation of the neuron. The mathematical representation of calculating the output of the neuron is shown below:

$$O_i = g(\sum_{\mathrm{j}=1}^{N}(w_{ij}x_j + b_i) \tag{2.1}$$

Where w is the learnable weights, b is the bias, x is the input and N is the number of input neurons. The function g() is an activation function to introduce non-linearity. The purpose of using weights w_{ij}, is to define how much a current node/neuron should listen to its input neurons. Here, bias b is also a learnable parameter. The process of calculating output from an input is called forward propagation. The way of updating the learnable parameters(weights and bias) is called backpropagation, which will be discussed in the next section.

2.2.1 Back Propagation

As explained in section 2.2, given the input to the ANN, the network outputs the prediction, which should be verified against ground truth and learn-able parameters(weights and

bias) should be updated until the difference between prediction and ground truth reaches an acceptable level. This process of updating the learnable parameters is called back-propagation in a training phase. A simple error function is the mean squared error:

$$E(X, \theta) = \frac{1}{2N} \sum_{\mathrm{j}=1}^{N} (\tilde{y}_{\mathrm{i}} - y_{\mathrm{i}})^2 \tag{2.2}$$

where, $\tilde{y}_{\mathrm{i}}$ is a desired output and y_i is the network output. N is the number of outputs. The loss function will be minimized by taking the gradient of the loss function with respect to the learnable parameters. First, the gradient of the loss function with respect to the final layer is calculated and the gradient of the loss function with respect to the first layer is calculated last. Hence the name back-propagation. A mathematical representation of the weights update is shown below.

$$\theta = \theta - \eta \nabla J(\theta) \tag{2.3}$$

where θ is the parameter to be updated, η is a learning rate, and $\nabla J(\theta)$ is the gradient of the loss function with respect to the parameter which will be updated. Here, choosing the optimal value for the learning rate will also have a positive impact on optimizing the error function.

Finally, when the optimal value for all the learnable parameters is available, which minimizes the loss function, it might be possible that the network just memorizes the features in the training set and perform badly on the validation and testing set. This problem is also known as over-fitting.

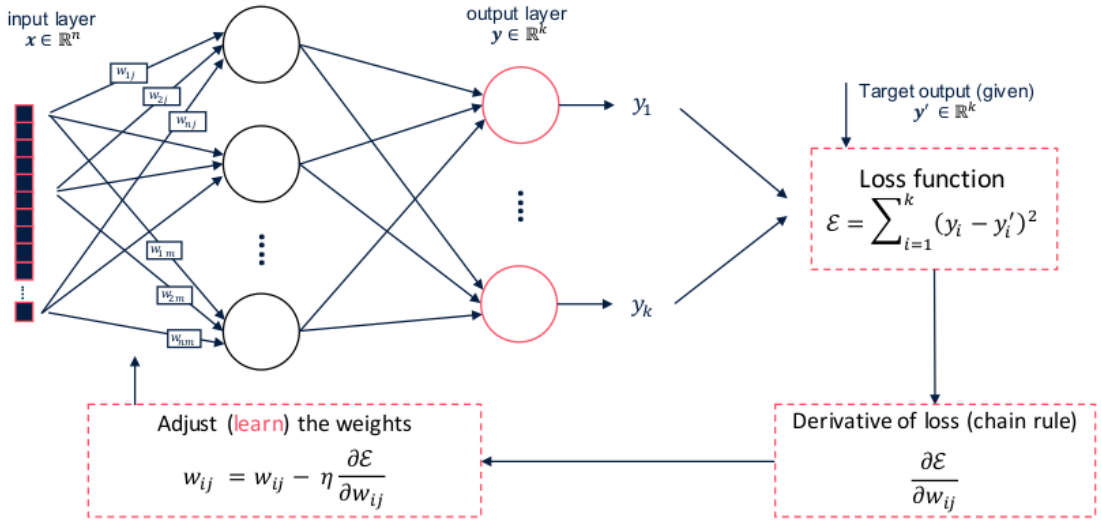


Figure 2.5: The schematic representation of back-propagation [Olof Berg Marklund 21].

2.2.2 Activation Functions

Choosing the right activation function, depending upon the neural network model, is an important design concept in DL. The decision of using a particular activation function in

the hidden layer defines how well the model learns the features from the training data-set and, also, in the output layer what the model predicts. Another important use case of using activation functions is to introduce non-linearity in the model. There exist different types of activation functions, which will be used in different parts of the model, such as the input layer, hidden layer, and output layer. The different activation functions alter the neural network model performance and capability. In this section, we will discuss different activation functions.

Sigmoid

The sigmoid function is commonly used in logistic regression classification problems. Where function takes real values as input and outputs a value between 0 and 1. If the value is positively large, the function outputs 1, whereas, for value towards more negative, the function outputs 0. The mathematical equation of sigmoid is given by:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

The problem with sigmoid activation function is, minimal updates of weights in the last layers, due to the reason of small gradients in the first layers during back propagation. This makes the network to take longer time to learn. This problem is called vanish gradients.

Tanh

The tanh stands for hyperbolic tangent activation function, similar to sigmoid, but outputs a value between -1 and 1. The output of tanh is 1, as the input is closer to positive and -1, as the input is closer to negative. The mathematical representation of tanh is given by:

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.5}$$

(a) Sigmoid **(b)** Tanh

Figure 2.6: Sigmoid and Tangent Hyperbolic Activation functions [Wikipedia 21 a].

Vanishing gradients problem still exits in the hyperbolic tangent activation function.

ReLU

ReLU stands for rectified linear activation function, most commonly used activation function. ReLU is easy to implement and effective against vanishing gradients as well. The ReLU outputs the value 0, if the input is negative, else input value is returned.

$$f(x) = max(0, x) \tag{2.6}$$

However, ReLU units are vulnerable to die during training. For instance, a large gradient passing through a ReLU neuron might cause the weights to be updated where the neuron will never activate again on any data point. If this occurs, the gradient flowing through that neuron will be zero from that point on. This is less problematic when the learning rate is set properly [Karpathy 21].

Leaky Relu

One solution to the "dying Relu" to use leaky Relu. Leaky Relu function outputs a small positive value, when the x $<$ 0, instead of 0. The mathematical representation of leaky Relu is given by:

$$f(x) = max(0, x) - \alpha * max(0, -x) \tag{2.7}$$

(a) Relu **(b)** Leaky Relu

Figure 2.7: Relu and Leaky Relu Activation functions.

2.2.3 Optimizers

The minimization of the loss function in a neural network is an important tuning process to obtain good accuracy. This process is an optimization problem. The optimization will be done by using gradient descent.

The gradient descent is a course of action taken to reduce an objective function $J(\theta)$ concerning the model parameters $\theta \in \mathbb{R}^d$ through updating the parameters in the direction opposite to gradient of objective function $\nabla J(\theta)$. Based on the amount of data used to calculate gradient descent of objective function, there exists a trade-off between the amount of time taken to update the learnable parameters and the accuracy. Below, three types of gradient descent will be discussed based on the above assumption.

Batch gradient descent

In batch gradient descent, we compute the gradient descent of error function $J(\theta)$ w.r.t learn-able parameters for the entire training data-sets, which results in slow update-rule and often leads to a memory issue on large training data-sets. The mathematical representation of batch gradient descent is shown below.

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{2.8}$$

Stochastic gradient descent

In contradiction to batch gradient descent, stochastic gradient descent update the learn-able parameters for each training example $x^{(i)}$ and $y^{(i)}$.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.9}$$

Instead of computing the same gradient descent for large data sets like in batch gradient descent, stochastic gradient descent(SGD) overcomes this repetition by computing the gradient descent for each example. Hence, SGD is faster and can be used for online learning(on the fly). However, it comes with a cost of heavy fluctuation of the objective function with high variance of parameters update as shown in the Figure 2.8 [Ruder 21].

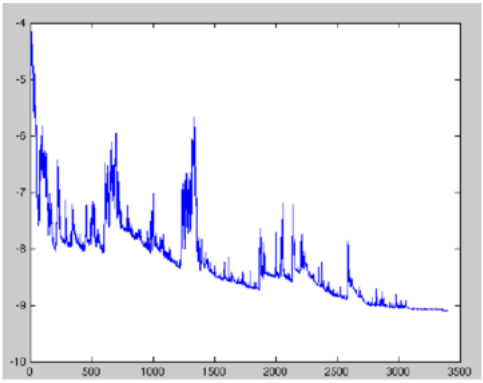


Figure 2.8: The large fluctuation of Stochastic Gradient Descent [Wikipedia 21 b].

Mini-batch gradient descent

The drawbacks of batch and stochastic gradient descent are overcome by mini-batch gradient descent by computing the gradient descent for n number of training examples.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.10}$$

The advantages of mini-batch gradient descent mainly lie in reducing the parameters update with high variance and solving the memory issue.

Nevertheless, mini-batch gradient descent holds some problems with it[Ruder 21].

- Challenges in choosing the optimal learning rate. A low learning rate leads to slow convergence and high learning rate results in fluctuation around the local minimum.

- Also, use of same learning rate for updating all parameters for every mini-batch/epoch. What if the data is sparse like in LiDAR, where parameters update should be depending on frequencies of feature occurrence, where rare occurrence needs a larger update.

The above-mentioned challenges are addressed using the SGD with momentum.

Momentum

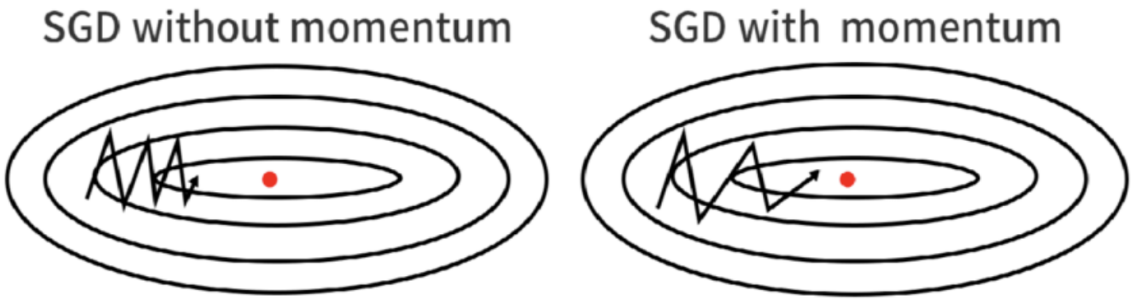


Figure 2.9: The behaviour of SGD with and without momentum [Du 19].

When the surface curves are steeper, SGD hesitates to move towards local minima and oscillates more. This obstacle is defeated by adding term momentum, which helps SGD to stimulate in the right direction and weaken the oscillation effect as shown in the Figure 2.9. The mathematical equation of SGD with momentum is:

$$\begin{aligned} v_t &= \gamma v_t + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{2.11}$$

Where, v_t is a update vector.

The idea of using momentum with SGD is similar to dropping a ball from the top, as it rolls down, the ball gets momentum until resistant from the air. The momentum increases as the gradient is direct in the same direction and decreases as the gradient points in the opposite direction results in faster convergence and reduced oscillation.

2.2.4 Batch Normalization

The difficulty in training a neural network grows as the number of layers increases. The challenge arises from the fact that after each mini-batch, when the weights are updated, the distribution of inputs to deep layers in the network may shift. As a result, the learning algorithm will continue to chase the target indefinitely. The shift in the distribution of inputs to layers is referred to as "Internal Covariate Shift" [Brownlee 21].

Batch normalization is a technique for training very deep neural networks in which the inputs to the layer are standardized for each mini-batch. The term "standardization" refers to a gaussian distribution with a mean of zero and a standard deviation of one. Batch normalization stabilizes the learning algorithm while also lowering the number of epochs needed to train the neural network [Brownlee 21].

The batch normalization also provides the effect of regularization, commonly used to reduce the generalization error. Hence, it is better not to use dropout with batch normalization.

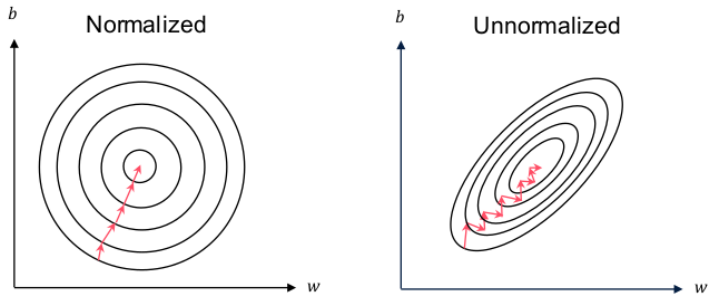


Figure 2.10: The gradient desecnt on normalized and unnormalized data. Path to optimum is easy and smooth for gradient descent on normalized data as compared to unnormalized data. [Olof Berg Marklund 21].

2.2.5 Pooling

The existence of features in an input image is summarized by convolutional layers in a convolutional neural network. But, the output feature map is sensitive to the location of features in the input. This issue is addressed by down-sampling the output feature map(reducing the width, height). The resulting down-sampled feature maps are more resilient to changes in the position of the feature in the image, which is referred to as "local translation invariance" in technical terms.

The pooling layer is an operation to down-sampling the feature maps by summarizing the existence of features. There are two common types of pooling, namely average pooling and max pooling. The average pooling summarizes the average presence of features, whereas max-pooling summarizes the most simulated features.

One practical use case of using pooling layers is, irrespective of the position of the phone either in our hands or in front of the chest while tagging the phone to our neck, still, classification algorithm should learn the phone features and classify them correctly.

Max Pooling

Given the 4x4 input feature map from a convolutional layer, max-pooling layers calculate the largest value in each patch of the input feature map. The max-pooling operation neglects the least features from the input feature map by keeping the most relevant features for the next convolutional layers. The visual representation of max-pooling is shown in Figure 2.11.

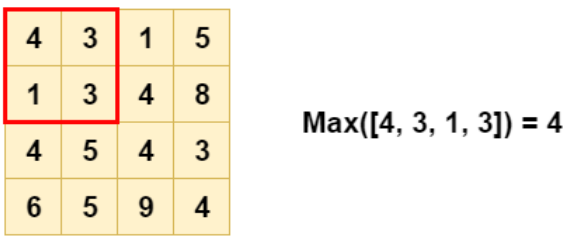


Figure 2.11: The visual representation of max pooling layer [Versloot 21].

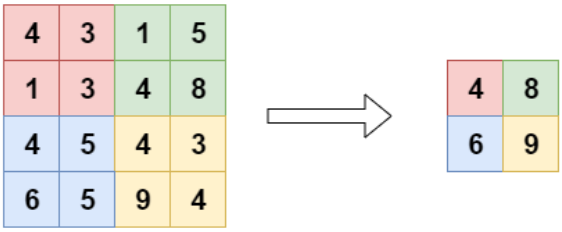


Figure 2.12: The output of convolutional layer after applying max pooling [Versloot 21].

Max pooling is more common to use as compared to average pooling. Max pooling keeps the features of the object of interest rather than averaging over unimportant features.

Average Pooling

As explained above, average pooling averages over the features, which retrains the information of less relevant features to the next convolutional layer.

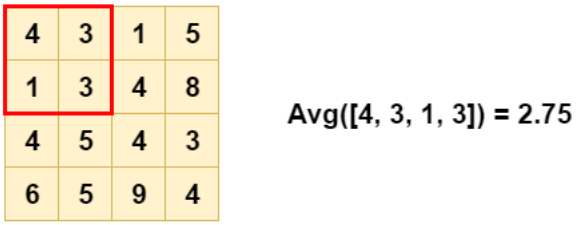


Figure 2.13: The visual representation of average pooling layer [Versloot 21].

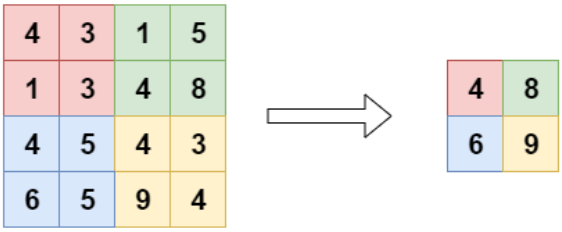


Figure 2.14: The output of convolutional layer after applying average pooling[Versloot 21].

1x1 Convolution

Another way of doing pooling is via 1x1 convolution(channel-wise pooling), which decreases or increases the number of feature maps. The main purpose of decreasing the size of the feature maps is to reduce the model complexity(large space and time complexity) as the networks get bigger, while retaining the salient features.

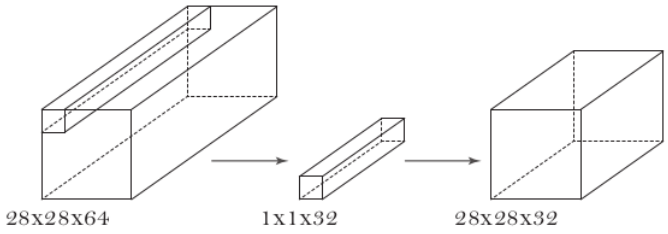


Figure 2.15: Reduction of number of feature maps using 1x1 convolution[Versloot 21].

2.2.6 Pruning

The generated bounding boxes from the object detection model either 2D/3D were subjected to pruning. The most commonly used pruning approach is non-max suppression(NMS). The purpose of doing pruning is to keep the most likely bounding box around the object. The NMS will be done in two stages.

- Based on the predicted classification score for each bounding boxes by the model
- Using intersection over union(IoU)

Intersection Over Union

The IoU computes the overlap between two shapes. The IoU will be done by computing the area of overlap over the area of a union as shown in the Figure 2.16.

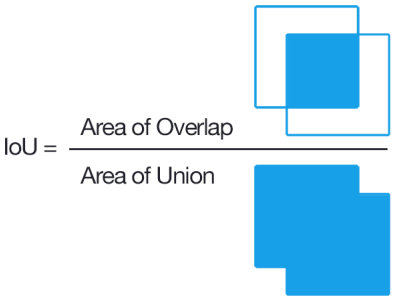


Figure 2.16: The calculation of IoU between two bounding boxes.

Typically, used in two different scenarios. During training, proposed bounding boxes are matched against ground truth, which strives for predictions. Secondly, IoU is also used as an evaluation matrix.

2.3 Convolutional Neural Networks

Convolution

The working principle of the convolutional neural network(CNN) is similar to the ANN/MLP as explained in section 2.2, because CNN is a special type of ANN. The Figure 2.17 illustrates the connection of neurons in CNN into neurons in FC layer. The neurons in

MLP receives input from all the neurons from the previous layer or input, where neurons in CNN receives input from a small patch of input. So, kernel/weight matrix moves along both directions(columns and rows) with specified strides and passes the result into an activation function. The number of kernels is equal to number of input channels. In CNN, shared weights across all the neurons in the same layer reduces the number of embedded parameters and makes the network to be more efficient. Below, different types of convolutional neural networks are discussed.

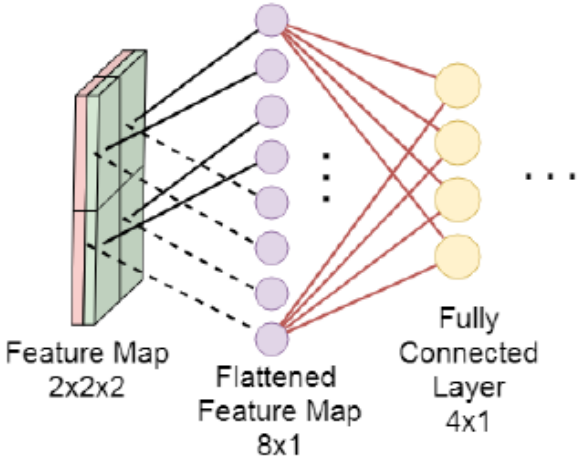


Figure 2.17: An illustration of connection of convolutional feature map 2X2X2 into fully connected layer of 4 neurons. Note that red lines are learnable parameters[Olof Berg Marklund 21].

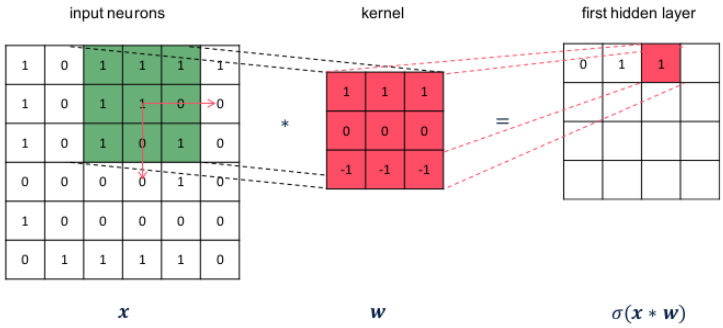


Figure 2.18: The convolutional layer takes input with 1 channel and applies convolution operation with a 3x3 kernel, stride=1, padding=0 [Olof Berg Marklund 21].

The mathematical expression for CNN with stride 1, a 3x3 kernel and zero padding is expressed as:

$$F_{n,m}(x;w) = \sigma((x * w)_{n,m}) = \sigma(\sum_{\mathrm{k=1}}^{C} \sum_{\mathrm{i,j=-2}}^{2} w_{i,l}^{k} \dot{x}_{n+i,m+j}^{k}) \qquad (2.12)$$
$$n = 1, ..., N \quad m = 1, ..., M$$

where $x \in \mathbb{R}^{NxMxC}$ is an input matrix with C channels and $w \in \mathbb{R}^{3x3xC}$ is a weight matrix.

2.3.1 2D Convolutional Neural Networks

2D CNN is mainly used on image data. Given RGB data of size 32x32x3 and filter/kernel of size 5x5x3 with stride 1, the resulting feature map size would be 28x28x1 after the filter sliding over the entire 2D image by taking the dot product on each patch of the image. The mathematical equation for 2D CNN is given by:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) * input(N_i, k) \tag{2.13}$$

where * is the valid 2D cross-correlation operator.

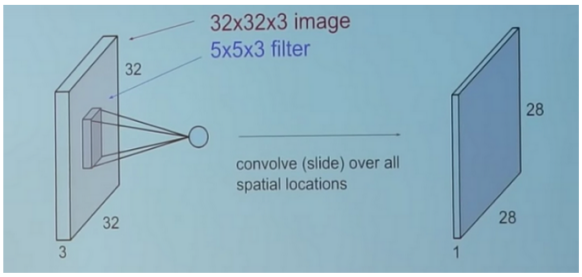


Figure 2.19: The convolutional layer takes RGB image data with 3 channel and applies convolution operation with three 5x5 kernel, stride=1, padding=0 [Olof Berg Marklund 21].

The calculation of output size $(N, C_{out}, H_{out}, W_{out})$ given the input size $(N, C_{in}, H_{in}, W_{in})$ with defined kernel size, padding, dilation and stride is mathematically described as:

$$\begin{aligned} H_{out} &= [\frac{H_{in} + 2 \times padding[1] - dilation[1] \times (kernel_size[1] - 1) - 1}{stride[1]} + 1] \\ W_{out} &= [\frac{W_{in} + 2 \times padding[2] - dilation[2] \times (kernel_size[2] - 1) - 1}{stride[2]} + 1] \end{aligned} \tag{2.14}$$

2.3.2 3D Convolutional Neural Networks

In 3D CNN, kernel slides over 3D data like data from 3D vision sensors or point cloud from LiDAR. The visual representation of 3D CNN is shown in Figure 2.20.

The mathematical equation for 3D CNN is given by:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) * input(N_i, k) \tag{2.15}$$

where * is the valid 3D cross-correlation operator.

For a given 3D input $(N, C_{in}, D_{in}, H_{in}, W_{in})$, output will be $(N, C_{out}, D_{out}, H_{out}, W_{out})$ Where N is the batch size, C_{in} is the feature vector, D_{in} is a depth in Z axis, H_{in} is a height in Y axis, and W_{in} is a width in X axis

The calculation of output size given the input size with defined kernel size, padding, dilation and stride is mathematically described as:

$$
\begin{aligned}
D_{out} &= [\frac{D_{in} + 2 \times padding[0] - dilation[0] \times (kernel_size[0] - 1) - 1}{stride[0]} + 1] \\
H_{out} &= [\frac{H_{in} + 2 \times padding[1] - dilation[1] \times (kernel_size[1] - 1) - 1}{stride[1]} + 1] \\
W_{out} &= [\frac{W_{in} + 2 \times padding[2] - dilation[2] \times (kernel_size[2] - 1) - 1}{stride[2]} + 1]
\end{aligned}
\tag{2.16}
$$

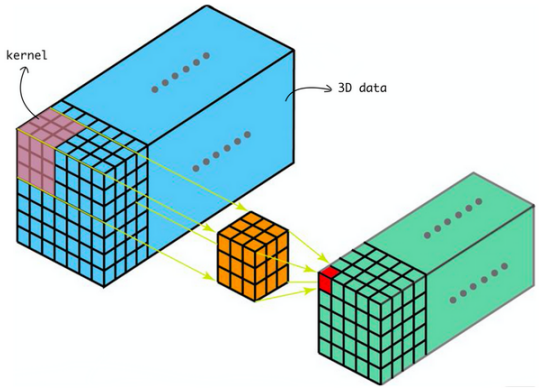


Figure 2.20: The ouput of 3D convolutional layer when a kernel slides over 3D data with stride=1, dilation=1, and padding =0[Pokharna 21].

The computational time for forward pass and backward pass is more as compared to 2D CNN along with number of embedded parameters.

3. Related Work

3D object detection models with LiDAR are greatly inspired by the enormous success of 2D object detection models using DL. Most of the 3D object detection models follow the same working principle of SOTA 2D object detection models.

In this section, a brief overview of current SOTA methods both in 2D and 3D object detection based on DL will be discussed.

3.1 2D Object Detection

The 2D object detection methods are broken down into two-stage detectors and single-stage detectors. In a two-stage detector, the first stage is used for feature extraction and proposal generation. While second-stage is used for refinement of proposed bounding boxes and class probabilities.

Unlike a two-stage detector, in a single-stage detector, there is no fully connected(FC) layer, where each features will have its own weights, after the proposal generation for further refinement of bounding boxes and class probabilities. The accuracy of single-stage detectors is less compared to a two-stage detector, but high inference time and better suitable for real-time applications.

In this section, the fundamentals of Faster R-CNN[Ren 15], a two stage detector and "Single shot multibox detector(SSD)" [Liu 16], a single stage detector will be explained.

3.1.1 Faster-RCNN

As explained earlier, Faster-RCNN is a two-stage detector, which means it has two networks: Region Proposal Network(RPN) for generating region proposals and another network for detecting the objects using the generated proposals. The distinction between Faster-RCNN and Fast-RCNN [Girshick 15] is, former uses the RPN to generate the proposals and later uses a selective search algorithm. Since the RPN shares the computation with the whole object detection network, so time for proposal generation is much less compared to the selective search algorithm.

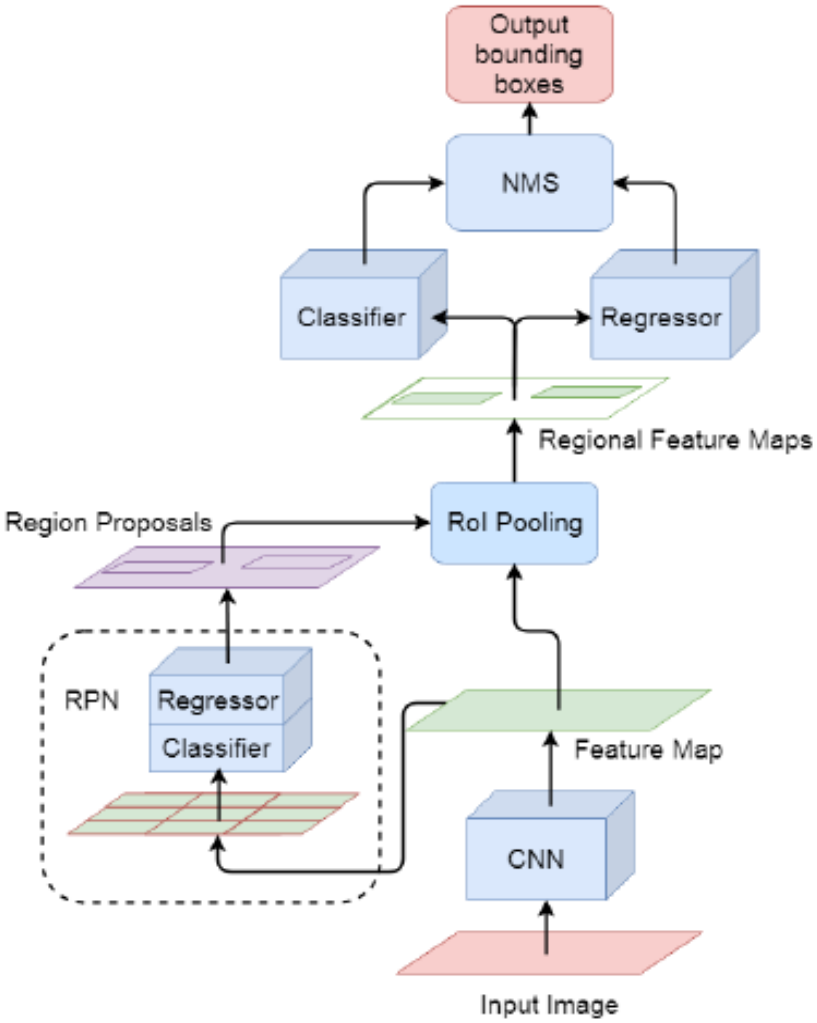


Figure 3.1: The architecture of Faster-RCNN [Olof Berg Marklund 21].

The input is tensor (Height x Width x Depth), passed through a collection of convolutional layers or pre-trained convolutional neural networks like VGG [Simonyan 14] and ResNet [He 16]. Then the output of the feature extractor would be a convolutional feature map. Later the convolutional feature map will be the input to the RPN, which proposes the bounding boxes for each spatial location. Here, RPN takes anchors as a reference to generate the bounding boxes over the image and anchors may be of different sizes and aspect ratios. Finally, RPN gives two different outputs, one is objectiveness score and the other is bounding box position to adjust the anchor to fit the predicted object's position. The RPN does not care about the object type, i.e., whether the object is pedestrian or cyclist. RPN just classifies between background and foreground [Ren 15].

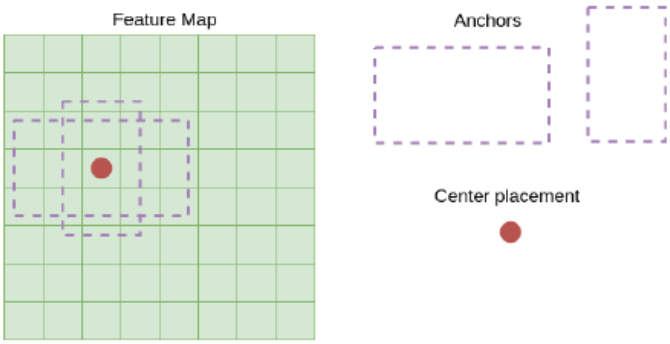


Figure 3.2: Placement of anchors at each spatial location on feature map [Olof Berg Marklund 21].

Next, only a limited number of proposed bounding boxes are taken into the next level, based on the percentage of overlap with ground truth as explained in section 2.2.6 during training. Considering the bounding boxes which are taken into the next level, the proposed bounding boxes overlap more than 50% and less than 10% with ground truth are classified as foreground and background respectively. While testing, proposed bounding boxes are limited based on the classification score, this score is referred to as the RPN threshold. Then the chosen bounding boxes are projected back to the convolutional feature map to get the feature maps. One thing to notice here is, size of the feature maps of chosen bounding boxes should be the same, this would be done by region of interest pooling(RoI) [Ren 15].

Finally, the output of RoI pooling is input to fully connected(FC) layers, then FC is connected to classification and regression loss function for further refinement of bounding boxes regression and classification prediction [Ren 15].

3.1.2 SSD

In the SSD model, e.g. VGG or ResNet is used as a 2D backbone for feature extraction like in Faster-RCNN. Then, similar to RPN in Faster-RCNN, SSD places the anchors of different sizes and aspect ratios on multi-scale convolutional feature maps at each spatial location. Thus, yielding the WxHxk bounding boxes for each feature map, where k is the number of anchors placed on the feature map of width W and height H.

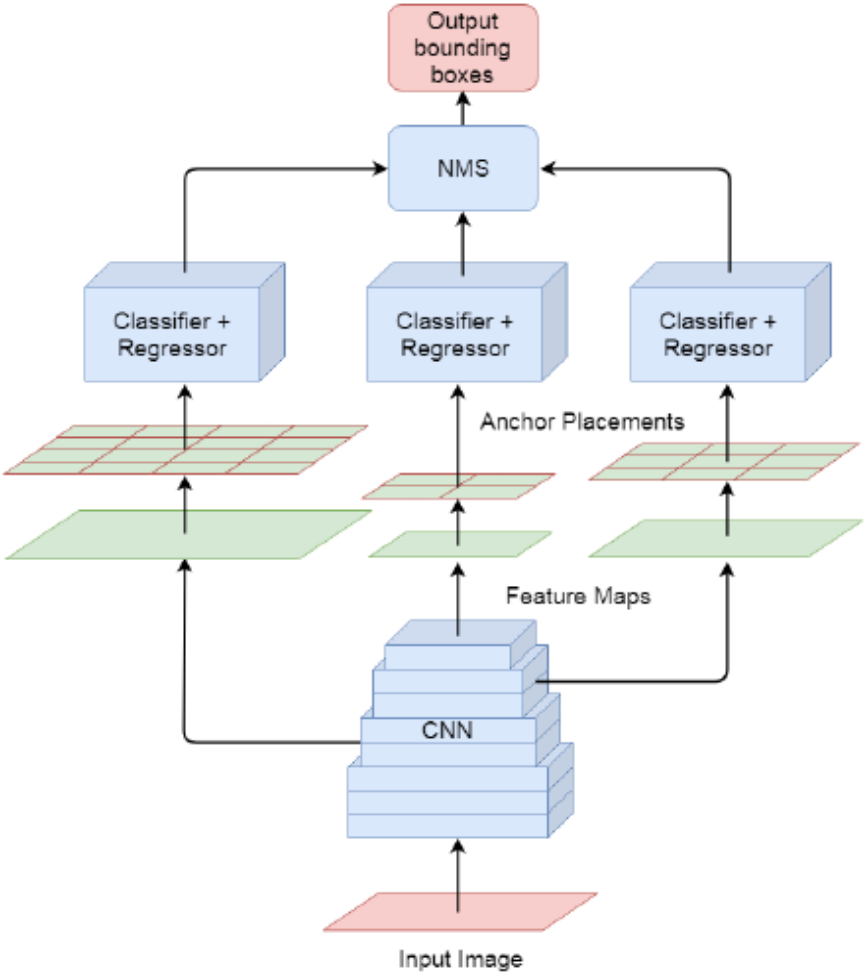


Figure 3.3: The simplified architecture of SSD [Olof Berg Marklund 21].

The bounding boxes in each feature map are matched against ground truth using the IoU approach as explained in section 2.2.6. If the percentage of overlap is more than a certain threshold, those bounding boxes are considered. Since the anchors are placed over multi-scale feature maps, there would be multiple bounding boxes for each object. So, those redundant bounding boxes are removed as explained in 2.2.6 [Liu 16].

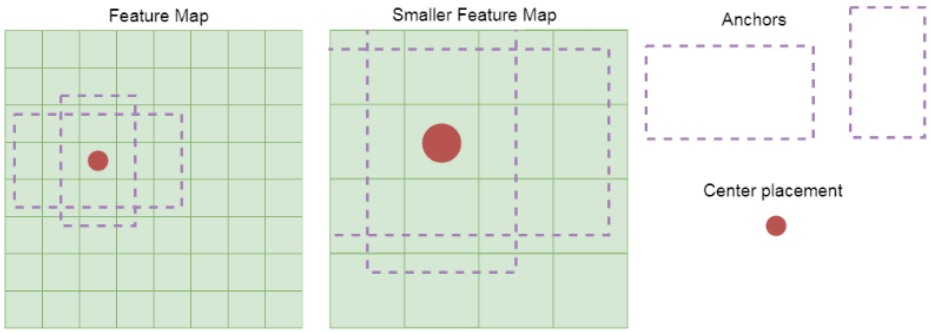


Figure 3.4: Placement of anchors on multi-scaled feature maps [Olof Berg Marklund 21].

3.2 3D Object Detection

As stated in 1.2, the properties of 3D data from LiDAR, mainly sparse and unstructured, which means the non-existence of points in most of the positions in space and invariant to shuffling of points. To handle this challenge, many of the deep learning-based 3D object detection models, mainly [Simony 18] converted the 3D data into bird's eye view(BEV) and then applied a 2D convolutional network. The bottleneck in the above-mentioned models is that information loss which results in failure of complete 3D feature extraction of an object [Zhou 18]. [Chen 17] converts point cloud into front view(FV) and BEV, then feature maps of FV and BEV combine with image feature map(late fusion), results in higher accuracy than BEV based methods, but perform poorly on small objects with increasing inference time because of three CNNs.

In considering the application of 3D object detection with LiDAR in AD, a robust LiDAR-only 3D object detection system should be included in the perception stack of AV. To overcome the drawbacks of the above-mentioned model, Qi et al. [Qi 17a] proposed PointNet which directly takes the point cloud and processes each point independently. Later, Shi et al [Shi 19] used PointNet++ [Qi 17b], an extension of PointNet, as a 3D feature extractor along with RCNN as a detection head for LiDAR only 3D object detection. Alternatively, Zhou et al. [Zhou 18] introduced the image-like representation of point cloud through voxelization, such that conversion of data into a regular format and feeding into the deep nets. In following the working principle of VoxelNet, PointPillars was developed by Lang et al [Lang 19]. The basic architecture of LiDAR-only-based 3D object detection model will be discussed in section 3.2.1 and 3.2.2.

3.2.1 PointRCNN

PointRCNN is a two-stage detection framework, which uses pointnet++ [Qi 17b], as a 3D backbone for feature extraction, where the first stage will generate a limited number of bounding boxes in a bottom-up manner through semantic segmentation. Particularly, 3D ground truth bounding boxes are used as a semantic mask for semantic segmentation. In the second stage, semantic features are extracted after doing point cloud region pooling and feature extraction of pooled point clouds after converting them into canonical coordinates from global coordinates. Finally, both the features are merged and bin-based regression loss is used for proposal refinement [Qi 17a].

3.2.1.1 Pointnet feature extractor

The pointnet is designed based on the principle of the output of the model should be invariant to permutations and transformations of an input point cloud. The architecture of pointnet is very simple and easy to understand. The pointnet takes point cloud of n points with 3 dimensions. Then each point undergoes geometric transformation on the working principle of [Jaderberg 15]. After each affine transformation, the dimension of each point will be increased from 3-64-128-1024 through a collection of convolutional 1D layers (Conv1D). The purpose of using the Conv1D layer is n number of points share the same weights and bias. Finally, max pooling is applied over the n points of 1024 dimensions. For the classification task, the output of max-pooling is connected to FC, then FC outputs m classification scores.

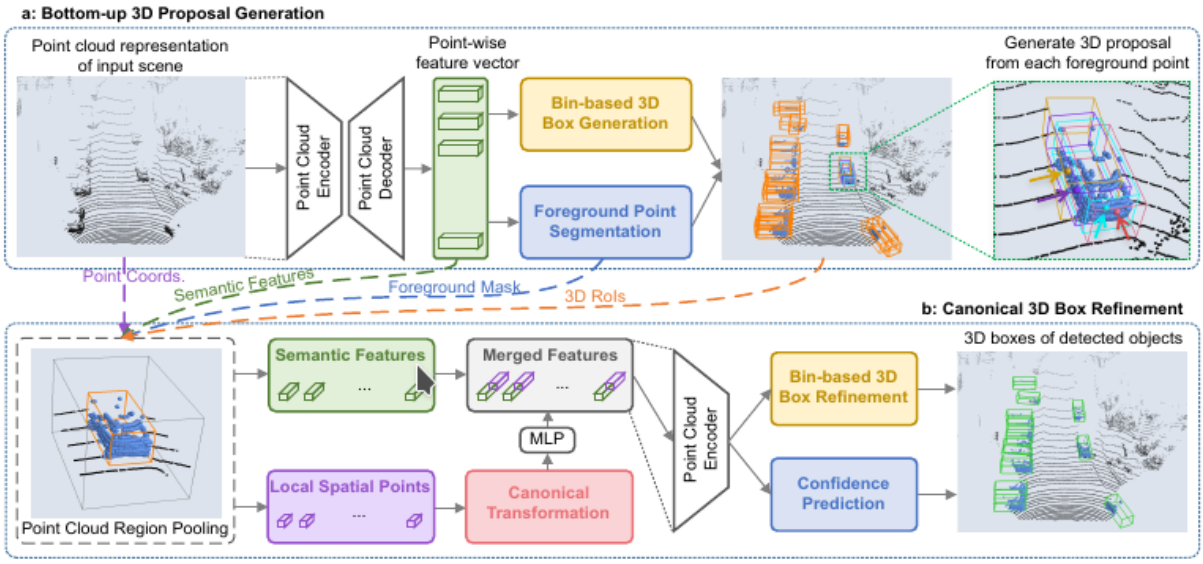


Figure 3.5: The network consists of two parts. (a) for generating 3D proposals from raw point cloud. (b) bounding boxes refinement in canonical coordinates [Shi 19].

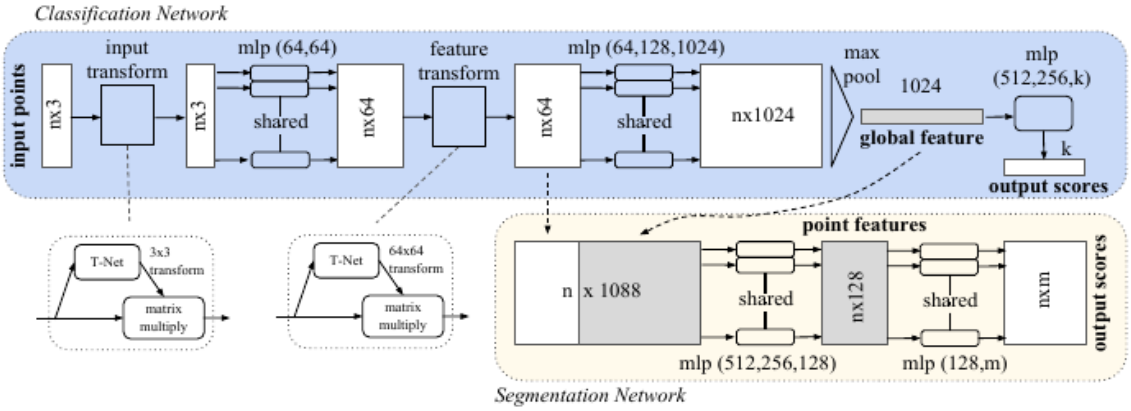


Figure 3.6: The pointnet architecture for feature extractor from point cloud. [Qi 17a].

The output of the semantic segmentation depends on both the local and global features. After the concatenation of both local and global features, n points with 1088 dimensions will be passed through pair of Conv1D. The output of the last Conv1D will be semantic segmentation scores for each point. Figure 3.6 depicts the architecture of pointnet.

However, pointnet does not extract the local/neighboring features, which results in low accuracy in classification and segmentation. To overcome this problem, Qi et, al.(2017) proposed pointnet++, which considers the local features into account. Pointnet++ consists of three abstraction levels: sampling layer(centroids of local regions), grouping layer(finding "neighboring" points around the centroids), and pointnet layer.

The sampling layers use the Iterative Farthest Point Sampling(FPS) algorithm to find the centroids. The centroids are a subset of input point clouds in each layer. Then in the grouping layer, using the ball query algorithm, n number of points will be grouped for each centroid. Here, n is a hyperparameter along with ball radius, and also n varies

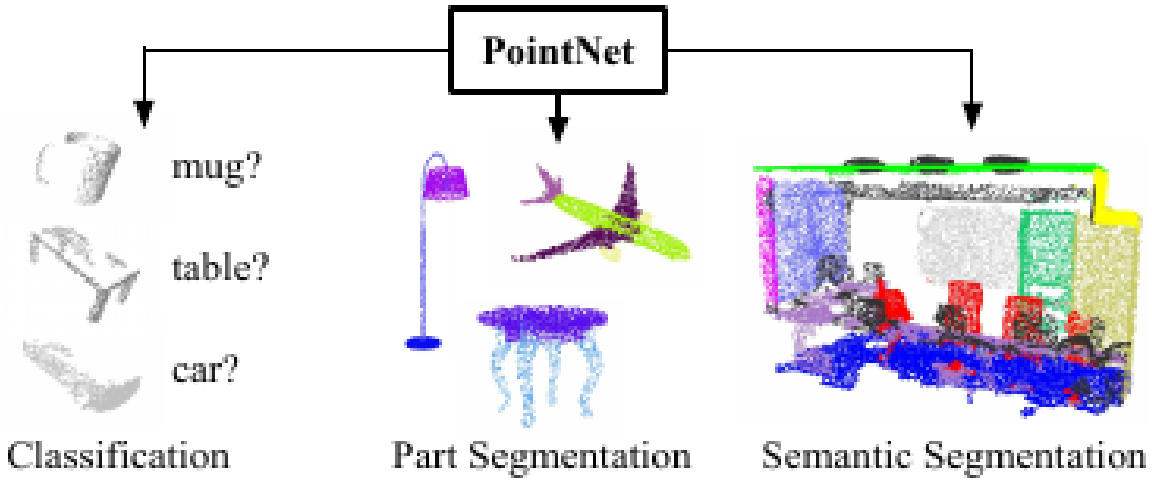


Figure 3.7: Classification and semantic segmentation of objects using pointnet. [Qi 17a]

based on the density of point clouds for each layer. Finally, pointnet is applied after each sampling and grouping layer, weights and bias will be shared among all the centroids.

For classification, the output of the final set abstraction layer will be passed through a collection of FC layers, which outputs classification scores.

For semantic segmentation, the output of the final set abstraction layer will be interpolated to get the feature of the previous layer, before applying the set abstraction layer(grouping, sampling, and pointnet). The interpolation is a kind of decoder. Then after a certain number of interpolations, the network outputs results of semantic segmentation for each point.

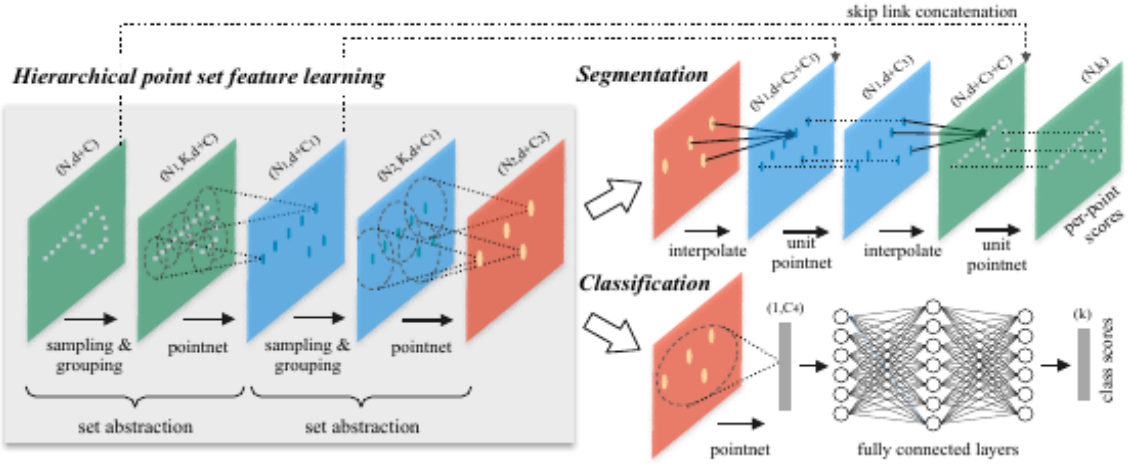


Figure 3.8: The architecture of pointnet++. [Qi 17b]

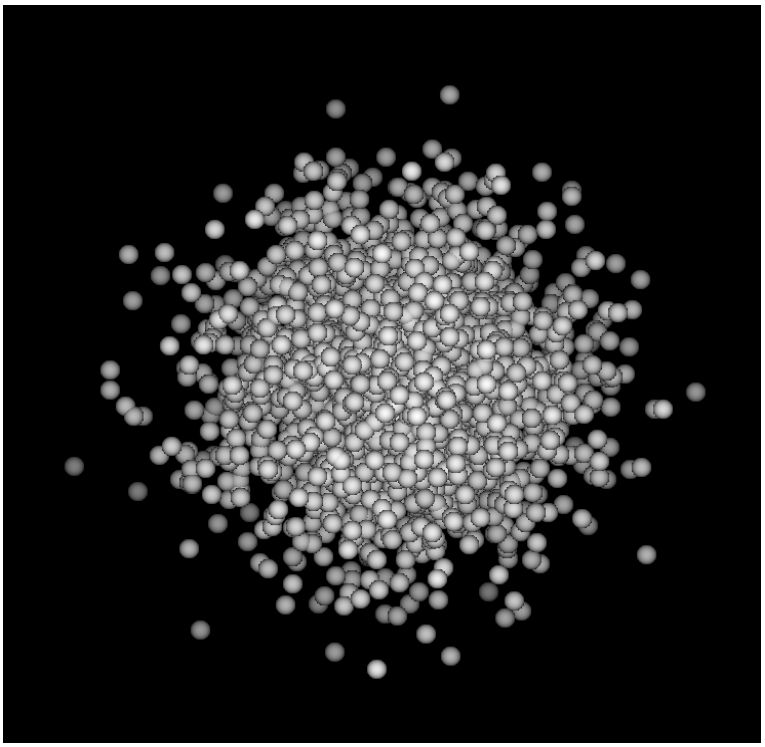


Figure 3.9: Simple representation of ball query method. [Qi 17b]

3.2.1.2 Bottom-up 3D proposal generation via point cloud segmentation

Pointnet++ is used to learn selective multi-scale point-wise features. It is also possible to use voxelnet as a 3D feature extraction backbone with sparse convolution. The availability of rich information in foreground points to predict their allied object's location and orientations, segmentation of foreground points will be constructive for 3D bounding box generation. The focal loss function is used to balance between the foreground and background points [Shi 19].

$$
\begin{aligned}
L_{focal}(p_t) &= -\alpha_t(1-p_t)^{\gamma}(log(p_t)) \\
p_t &= \begin{cases} p & \text{for foreground point} \\ 1-p & \text{otherwise} \end{cases}
\end{aligned} \tag{3.1}
$$

After the proposal generation, to learn more specific local spatial features, 3D region of pooling(RoI) is applied for each box $b_i = \{x_{\mathrm{i}}, y_{\mathrm{i}}, z_{\mathrm{i}}, l_{\mathrm{i}}, w_{\mathrm{i}}, h_{\mathrm{i}}, \theta_{\mathrm{i}}\}$. The RoI is accomplished by augmenting the constant value to height, width and length of box b_i. If the point still exists in the b_i, point and its features, it's segmentation mask are kept for the second stage [Shi 19].

3.2.1.3 Canonical 3D bounding box refinement

To benefit from the advantage of high-recall from stage-1, pooled points are transformed into a canonical coordinates system(CCS). Then features of points are extracted in CCS as well and concatenated with pooled point features for final prediction. The same regression

loss function is used for proposal generation as well as box refinement in CCS as proposed in [Shi 19]. The advantages and disadvantages of pointnet++ based models:

+ Better in accuracy as compared to voxelnet based models [Zhou 18]

+ Better in localization

+ Better feature extraction along with relationships between neighboring points

- Low inference time as compared to voxelnet based models

- Performance of the models depends on point cloud density

- Region proposal generation is not good as compared to voxelnet

- Inference time decreases as the density of point cloud increases

3.2.2 PointPillars

Since the object detection with LiDAR point cloud is a three-dimensional problem the use of 3D convolution is indispensable as in [Zhou 18], results in low inference time at 4.4 *Hz*; nevertheless, the performance of voxelnet is very good. To make use of a 3D object detection model in real-time on autonomous driving, Lang et al. (2019) developed a voxelnet based model to use 2D convolutional as a feature extractor/encoder.

The architecture of point pillars(PP) is very simple to understand and use as shown in Figure 3.10. It consists of point pillar feature extraction layer, followed by 2D convolution and SSD detect head.

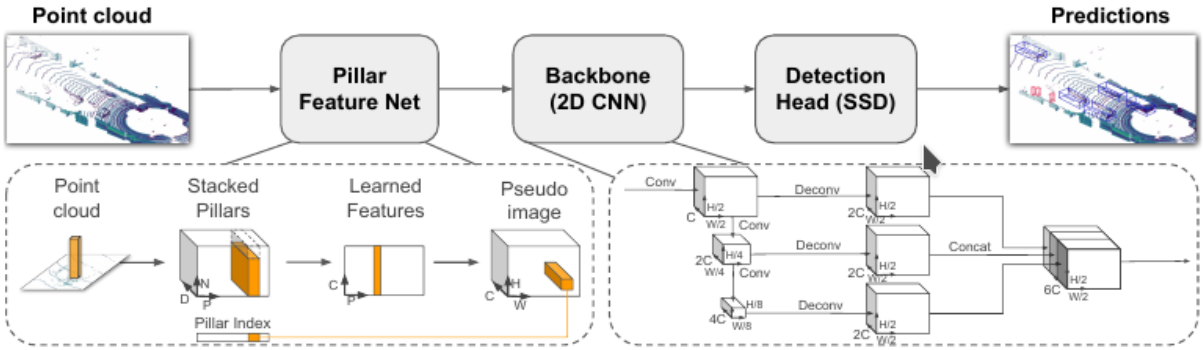


Figure 3.10: The network consists of two parts. (a) point feature extraction layer. (b) 2D backbone and SSD detect head [Lang 19].

3.2.2.1 Point pillar feature extraction layer

To make use of 2D convolution instead of 3D convolution, the point cloud should be converted into a pseudo image. Each point is represented by l in a point cloud with

coordinates x,y,z, and reflectance r. Initially, the whole point cloud should be discretized into a small number of bins in the xy plane, leading to a set of pillars P. Unlike in voxelnet, pillar height is not a hyperparameter. Then each point in a pillar is augmented with x_c, y_c, z_c, x_p and y_p where the c subscript represents distance to the arithmetic mean of all points in the pillar and the subscripts denotes offset from the pillar x,y center. Finally, the dimensionality of augmented LiDAR point became D=9 [Lang 19].

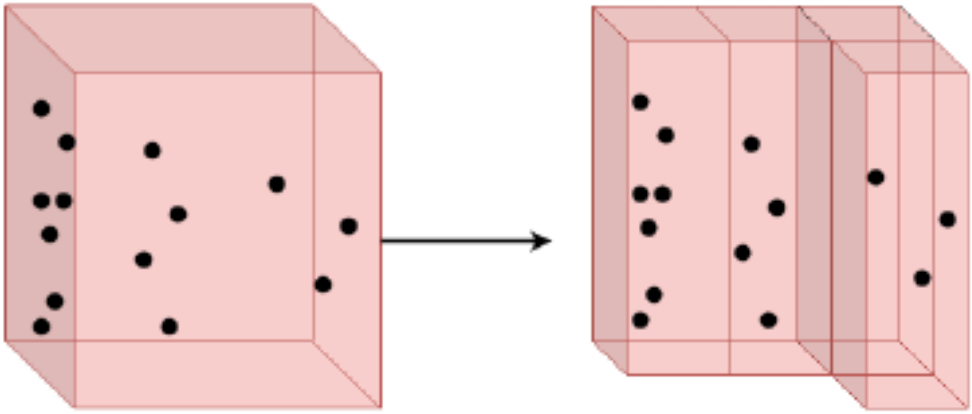


Figure 3.11: Simplified illustration of how point clouds are discretized into pillars. [Olof Berg Marklund 21].

However, most of the pillars are empty, due to the sparse nature of point cloud, the number of pillars is limited along with the number of points per pillar to construct a tensor of size (D, P, N). If the pillars hold more than threshold points, points are randomly sampled and if the pillars accommodate fewer points than the threshold, zero paddings are applied [Lang 19].

Then a simple version of pointnet is used, where a linear layer followed by batch normalization and ReLU to each point to generate a tensor of size (C, P, N). In the end, max-pooling operation over the channels to output the tensor of size(C, P). Finally, a pseudo-image of size (C, H, W) is created by scattering the features back to the location of the initial pillar [Lang 19].

The advantages and disadvantages of point pillar based models:

+ High inference time(60 Hz)

+ Accuracy is better than BEV and fusion-based models

- Accuracy is not good as compared to pointnet++ and voxelnet based models

- Performance degradation as the density of point cloud decreases

3.3 Data Annotation

The most important part of supervised learning is the ground truth to minimize the predicted loss of deep learning models through backpropagation. Unlike annotation of image data in 2 degrees of freedom(DoF), annotation of 3D data is very complex and tedious work. Following are the reasons:

- Annotation in 6 DoF
- Requires lots of computational power
- More disk usage either on-premises or on cloud
- Realistic annotation based on the human visual system
- Requires well designed and developed software

According to our research, none of the open-source libraries fulfill the above-mentioned requirements for 3D data annotation. Based on our research, we have found that the majority of the research institute(e.g. DFKI Kaiserslautern, Karlsruhe Institue of Technology(KIT), Technische Hochschule Ingolstadt) have developed an in-house tool for 3D data annotation for their research purposes. On the other hand, by seeing the huge business opportunity in 3D data annotation, many start-ups have been founded and a subsidiary of Amazon, amazon web services(AWS) also developed a 3D data annotation tool.

The crucial part among the 3D data annotation tools is, annotation of ground truth in 3D can be used to do the ground truth annotation of image data by using the transformation and projection matrices, which in turn reduce the time and cost for annotation of different sensors data individually.

Figure 3.12: AWS sagemaker used for annotation of LiDAR point cloud in 3D in 6 DoF and projecting the ground truth on image.

Apart from AWS, there are other open-source 3D annotation tools like latte [Wang 21] and the rest are based on latte itself. But latte does the annotation in BEV, then the height of the objects should be set by the annotators manually in the ground-truth data

file. There were also attempts to extend the latte as a 3D annotation tool, but most of the tools should not provide the support for the issues in using their tools. MATLAB also provides the 3D annotation tool, but annotation using MATLAB is too slow, complex, and not a user-friendly user interface.

All the above-mentioned tools are thoroughly tested along with different annotation tools from different start-ups.

The difficulties in testing the 3D annotation software are:

- No proper support for open source tools
- One has to read lots of documentation for paid 3D annotation softwares like AWS, MATLAB and other start-ups along with converting data into tool's specific format.

Recently, [Openvinotoolkit 21] released an open-source annotation tool for LiDAR 3D data in July 2021, but it is yet to get stable. [autonomousvision 21] also stated that the 3D annotation tool will be made public in near future.

4. VoxelNet

This chapter presents voxelnet [Zhou 18], an end-to-end model for 3D object detection with LiDAR point clouds without using any handcrafted features. The model takes the point cloud as an input and outputs the position of objects in 3D in a format of [x, y, z, h, w, l, θ] and the type of the object. Currently, voxelnet is one of the most widely adopted models as a 3D backbone. The followings are the reasons.

- Complete extraction of object features in 3D through voxelization
- Unlike [Simony 18] and [Chen 17], no loss of information during pre-processing
- No need for the calibration and synchronization between the camera and LiDAR like in [Ku 18]
- Better 3D region proposals generation as compared to pointnet and its variants
- Better inference time in the variants of voxelnet along with high accuracy, which makes it to be used in real-time in autonomous driving
- Performs well even with the low number of 3D points

4.1 Architecture

Voxelnet is a single-stage network, consisting of three layers mainly: Feature Learning Network, Convolutional Middle Layers, and Region Proposal Network as shown in Figure 4.1.

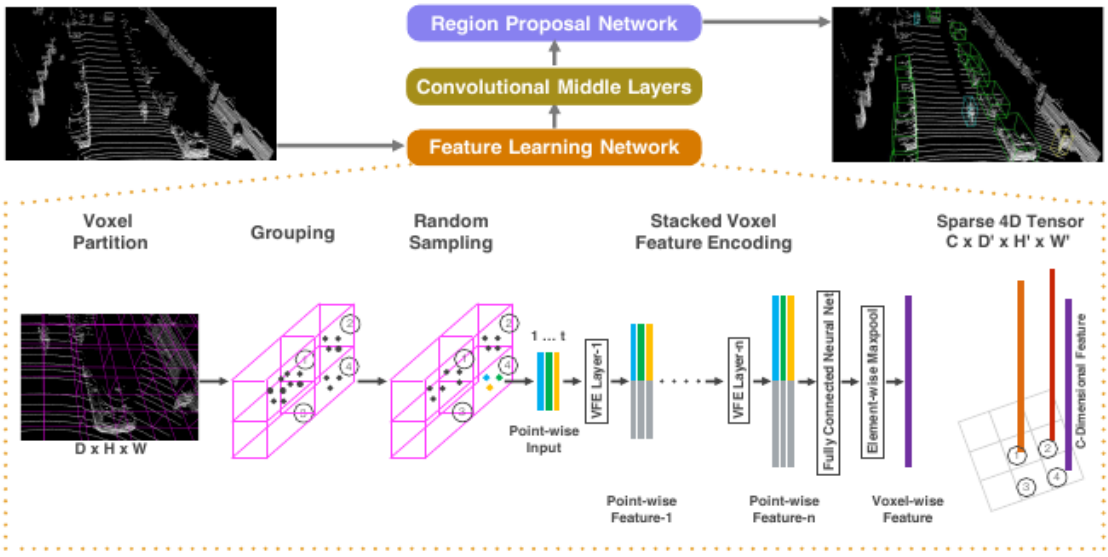


Figure 4.1: Voxelnet architecture. The feature learning network takes the raw point cloud as input and convert the point cloud into image-like representation through voxelization. Then the middle layers processes the 4D tensor, finally RPN generates the 3D proposals [Zhou 18].

4.1.1 Feature Learning Network

4.1.1.1 Point cloud pre-processing (Voxelization)

The point cloud from lidar scanner in 3D space are uniformly discretized into 3D voxel grid cell as presented in Figure 4.2. In the case of a point cloud in 3D space with ranges D, H, and W along Z, Y, and X, each cell of the voxel grid is assumed to be of size v_D, v_H, and v_W, respectively. Then the final voxel grid of size D´ = D/v_D, H´ = H/v_H, W´ = W/v_W [Zhou 18].

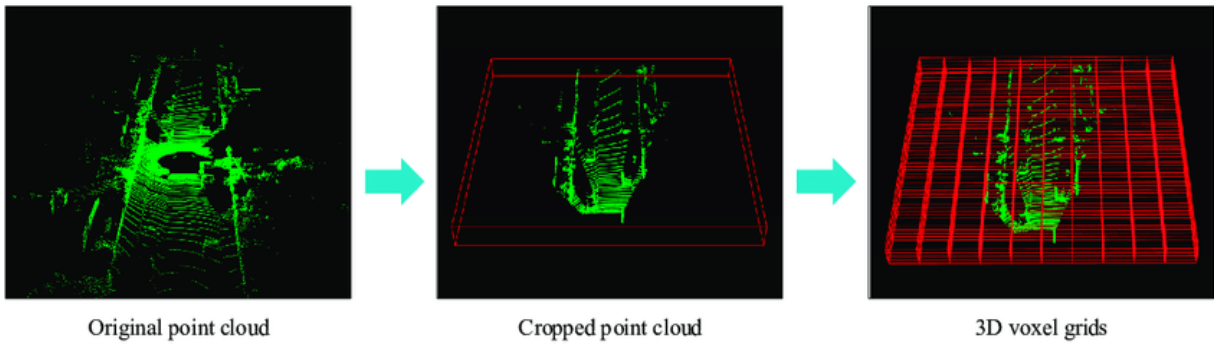


Figure 4.2: A simple representation of how point clouds are broken down into voxels [Xu 19].

The implementation of the voxelization algorithm in python is shown in next page. Depending upon the use cases, scene size, voxel size, grid size, lidar coordinate, and maximum point number variables should be adopted in the code.

```
def voxelization(point_cloud):
    # Input:
    #   (N, 4)
    # Output:
    #   voxel_dict

    scene_size = array([4, 40, 48]) #(Z,Y,X)
    voxel_size = array([0.4, 0.2, 0.2]) #(Z,Y,X)
    grid_size = array([10, 200, 240]) #(Z,Y,X)
    lidar_coordinate = array([0, 20, 3]) #(X,Y,Z)
    maximum_point_number = 45

    random.shuffle(point_cloud)

    shifted_coord = point_cloud[:, :3] + lidar_coord
    # reverse the point cloud coordinate (X, Y, Z) -> (Z, Y, X)

    #assigns the indices(x,y,z) to each point (multiple point
       can have same indices)
    voxel_index = floor(shifted_coord[:, ::-1] / voxel_size)
    #assigns the true/false(0/1) to each indices which full-fill
        the conditions
    bound_x = logical_and(voxel_index[:, 2] >= 0,
              voxel_index[:, 2] < grid_size[2])
    bound_y = logical_and(voxel_index[:, 1] >= 0,
              voxel_index[:, 1] < grid_size[1])
    bound_z = logical_and(voxel_index[:, 0] >= 0,
              voxel_index[:, 0] < grid_size[0])
    #choose the indices based on the below condition
    bound_box = logical_and(np.logical_and(bound_x, bound_y),
                bound_z)

    #finally select the points of true values
    point_cloud = point_cloud[bound_box]
    #select the indices of true values
    voxel_index = voxel_index[bound_box]

    # [K, 3] coordinate buffer as described in the paper
    coordinate_buffer = unique(voxel_index, axis=0)

    K = len(coordinate_buffer)
    T = max_point_number

    # [K, 1] store number of points in each voxel grid
    number_buffer = zeros(shape=(K))

    # [K, T, 7] feature buffer as described in the paper
```

```
    # feature buffer contains, voxel features, voxel indices,
       voxel number
    feature_buffer = zeros(shape=(K, T, 7))

    # build a reverse index for coordinate buffer
    index_buffer = {}
    for i in range(K):
        index_buffer[tuple(coordinate_buffer[i])] = i

    #group less than T number of points for each voxel
    #each voxel may conatins differnt points in it, from 1 to T
    for voxel, point in zip(voxel_index, point_cloud):
        index = index_buffer[tuple(voxel)]
        number = number_buffer[index]
        if number < T:
            feature_buffer[index, number, :4] = point
            number_buffer[index] += 1

  #dimensionality increase of each point from 4 to 7
  #centroid(sum of all points divided by number of points in a
     voxel) calculation and augment each point with centroid
    feature_buffer[:, :, -3:] = feature_buffer[:, :, :3] - \
        feature_buffer[:, :, :3].sum(axis=1, keepdims=True)/
           number_buffer.reshape(K, 1, 1)

    voxel_dict = {'feature_buffer': feature_buffer,
                  'coordinate_buffer': coordinate_buffer,
                  'number_buffer': number_buffer}
    return voxel_dict
```

Voxelization Implementation

The original code for point cloud pre-processing was implemented in python using the numpy library and the numba library was used to run voxelization code on GPU. To run the 3D object detection model on hardware, initially, voxelization was implemented using the Tensorflow[Google 15] computational graph concept. But the execution time was 17 seconds on CPU. Then, the point cloud pre-processing was implemented in C++ using the Eigen library. The execution time of pre-processing code was 350 milliseconds on CPU. But, to use the voxelnet based 3D object detection models in real-time, point cloud pre-processing should be implemented on GPU using CUDA programming language/Tensorflow framework. Following are the reasons:

- CPU is good for sequential operation, not for operation on matrices
- GPU is specifically designed to perform an operation on matrices
- Copying the pre-processed data from CPU to GPU is an expensive operation

Another important framework to work on point cloud is point cloud library(PCL) [open source 21]. PCL has the voxelization feature, which is implemented as an octree. But, octree provides the pointers to access the voxels, not the voxel's indices and number.

4.1.1.2 Grouping

After the voxelization, points are grouped where they reside in. Due to the sparse nature of point cloud, each voxel grid will hold extremely variable points in them. An example is shown in Figure 4.3, where voxel 2 has relatively fewer points than voxel 1, but more than voxel 3 and 4.

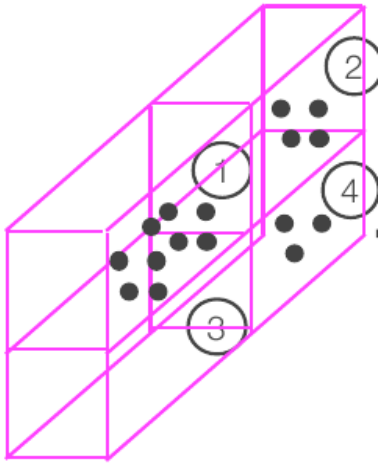


Figure 4.3: A simple representation of voxels in voxel grid [Xu 19].

4.1.1.3 Random sampling

Commonly, the number of points in a point cloud would be more than 100k, depending upon sensors, processing of all points demands more computation and time to train a model along with adding the bias to the final detection results. For that reason, only T number of points are sampled from each voxel grid cell, if they contain more than the T number of points. If not, just sample the available number of points [Zhou 18].

4.1.1.4 Stacked Voxel Feature Encoding

The introduction of series of voxel feature encoding(VFE) layers is the key innovation in voxelnet. The sequential feature encoding for one voxel is shown in Figure 4.1. Figure 4.4 represent the architecture of VFE Layer-1.

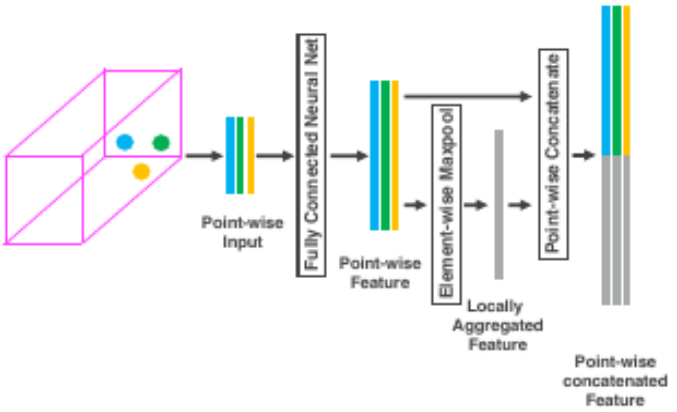


Figure 4.4: A simple illustration of voxel feature encoding for a voxel.

Considering V $= \{p_i = [x_i, y_i, z_i, r_i]^{\mathrm{T}} \in \mathbb{R}^4\}_{\mathrm{i=1..t}}$ as a non-empty voxel consists of $t \leq T$ LiDAR points, where p_i have XYZ coordinates of the i-th point and r_i is the received reflectance. Initially, centroid of all the points in V are calculated, represented as (v_x, v_y, v_z) and augment with each point p_i in V, then attained input feature set, $V_{in} = \{\hat{p}_i = [x_i, y_i, z_i, r_i, x_i - v_x, y_i - v_y, z_i - v_z]^{\mathrm{T}} \in \mathbb{R}^7\}_{\mathrm{i=1..t}}$. To encode the shape of the surface contained within the voxel, each $\hat{p}_i$ is passed through fully connected network(FCN) into a feature space. The FCN is comprised of a linear layer, a batch normalization(BN) and a rectified linear unit(ReLU). Later, element-wise max pooling is applied across all point-wise feature f_i corresponding to V to obtain locally aggregated feature $\tilde{f} \in \mathbb{R}^{\mathrm{m}}$ for V. Lastly, locally aggregated feature $\tilde{f}$ is concatenated with each point-wise feature f_i, which results in point-wise concatenated feature as ${f_i}^{\mathrm{out}} = [{f_i}^{\mathrm{T}}, \tilde{f}^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{2\mathrm{m}}$. Hence the resultant output feature set is $V_{out} = \{{f_i}^{\mathrm{out}}\}_{\mathrm{i=1..t}}$. Likewise, all non-empty voxels are encoded as described above[Zhou 18].

VFE-i(c_{in}, c_{out}) is employed to denote i-th VFE layer that converts input features of dimension c_{in} into output feature of dimension c_{out}. Multiple VFE layers are stacked to learn more descriptive shape information within the voxel, which allows points interaction within the voxel as well.

Since most of the voxels are empty, roughly around 90%. By processing only non-empty voxels, reduce the memory usage and computation cost during backpropagation in the feature learning network layer. Hence, each non-empty voxel is associated with spatial coordinates in 4D space.

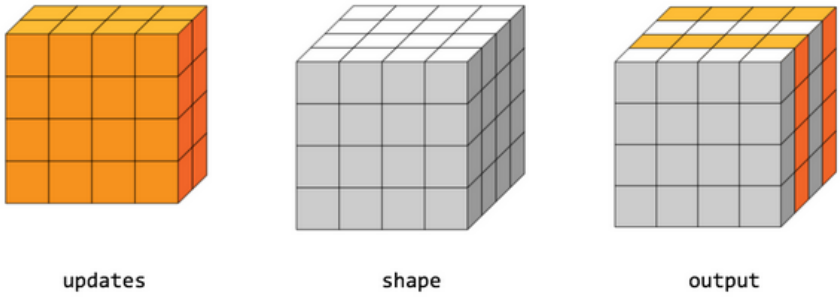


Figure 4.5: Building the 3D voxel grids using the Tensorflow scatter operation after the feature learning network.[Google 15].

As shown in the Figure 4.5, the forward and backward pass will be computed only for occupied voxels in the feature learning network, which in turn reduces the computational time. The input to the next convolutional 3D layer will be the output of scatter operation as shown in Figure 4.5.

4.1.2 Convolutional Middle Layers

ConvMD(c_{in},c_{out},k,s,p) is used to denote M-dimensional convolutional operator where c_{in} and c_{out} are the input and output channels, k, s and p are the M-dimensional vectors corresponding to kernel size, stride size and padding size respectively. We adopt a scalar to express the size when the size across the M-dimensions are the same e.g. k for k = (k,k,k). 3D convolution, BN layer, and ReLU layer are applied to each convolutional

middle layer, correspondingly [Zhou 18]. The filters sizes in middle convolutional layers will be explained in section 4.3.2.

4.1.3 Region Proposal Network

As explained in section 3.1.1, region proposal networks are vital for top-performing object detection models. After doing a certain change to the RPN proposed in [Ren 15], RPN is combined with feature learning network and middle convolutional layers to design an end to end trainable model [Zhou 18]..

The feature map from the convolutional middle layers is fed into an RPN layer as an input. Figure 4.7 illustrates the simple RPN architecture for multi-label 3D object detection. The RPN consists of three blocks of fully convolutional layers. The input feature map is down-sampled via the first layer of convolution with stride 2 and another convolution with stride 1 is applied, followed by BN, and ReLU operations after each convolutional layer in each block, respectively. the output of each block is up-sampled and high-resolution feature map is constructed through concatenation. Finally, the high-resolution feature map outputs the regression and probability score of desired targets [Zhou 18].

Single-class Architecture

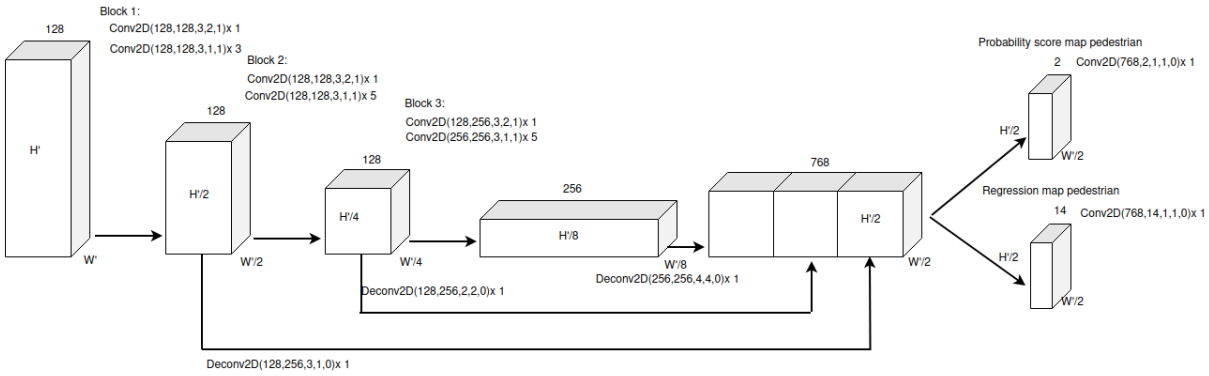


Figure 4.6: Region proposal network architecture for single-class 3D object detection.

Multi-label Architecture

Since the original model was designed for single-class object detection, the model is extended to work for multi-label 3D object detection, mainly pedestrian and cyclist. The below figure depicts the multi-label voxelnet.

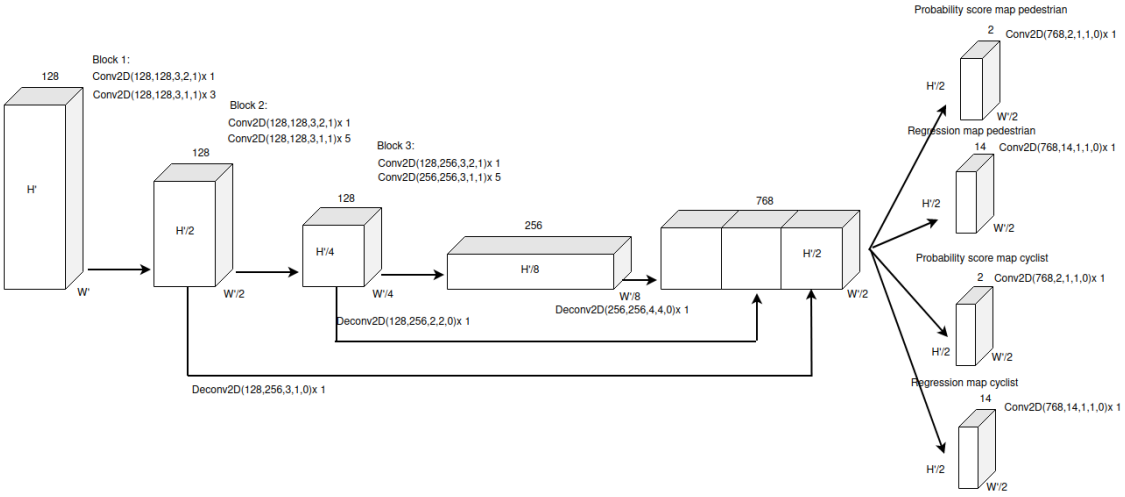


Figure 4.7: Region proposal network architecture for multi-label 3D object detection.

4.2 Loss Function

Loss functions are used to measure the performance of model prediction in terms of being able to predict the expected values. Commonly loss function is classified into classification and regression loss functions.

Classification

Classification refers to the process of categorization.

Commonly used classification loss function is cross entropy.

The cross entropy is defined mathematically as:

$$CE = -\sum_{\mathrm{i}=1}^{C} y_i * log(\hat{y}_i) \tag{4.1}$$

Where y_i is ground truth and $\hat{y}_i$ is the predicted classification score for each class i in C.

In this work, binary cross entropy loss function is used.

Binary cross entropy is defined mathematically as:

$$CE = -\sum_{\mathrm{i}=1}^{C=2} y_i * log(\hat{y}_i) = -y_1 log(\hat{y}_i) - (1 - y_i) log(1 - \hat{y}_i) \tag{4.2}$$

Regression

$\{a_{\mathrm{i}}{}^{\mathrm{pos}}\}_{\mathrm{i}=\,1\ldots\mathrm{N_{pos}}}$ and $\{a_{\mathrm{j}}{}^{\mathrm{neg}}\}_{\mathrm{i}=\,1\ldots\mathrm{N_{neg}}}$ denotes the set of N_{pos} (positive) and N_{neg} (negative) anchors, respectively. The 3D ground truth box is represented as $\{x_{\mathrm{c}}{}^{\mathrm{g}}, y_{\mathrm{c}}{}^{\mathrm{g}}, z_{\mathrm{c}}{}^{\mathrm{g}}, l_{\mathrm{c}}{}^{\mathrm{g}}, w^{\mathrm{g}}, h^{\mathrm{g}}, \theta^{\mathrm{g}}\}$ where $x_{\mathrm{c}}{}^{\mathrm{g}}, y_{\mathrm{c}}{}^{\mathrm{g}}, z_{\mathrm{c}}{}^{\mathrm{g}}$ are center location, $l_{\mathrm{c}}{}^{\mathrm{g}}, w^{\mathrm{g}}, h^{\mathrm{g}}$ are length, width and height of the box

and θ^{g} is the rotation around Z-axis. $\{{x_{\mathrm{c}}}^{\mathrm{a}}, {y_{\mathrm{c}}}^{\mathrm{a}}, {z_{\mathrm{c}}}^{\mathrm{a}}, {l_{\mathrm{c}}}^{\mathrm{a}}, w^{\mathrm{a}}, h^{\mathrm{a}}, \theta^{\mathrm{a}}\}$ represents the retrieved ground truth box after matching with positive anchors.

$u^* \in \mathbb{R}^7$ denotes residual vector which holds 7 regression targets corresponding to center location $\triangle x, \triangle y, \triangle z$, length, width, height as $\triangle l, \triangle w, \triangle h$ and the rotation $\triangle\theta$, which are calculated as:

$$\begin{aligned} &\triangle x = \frac{x_c^g - x_c^a}{d^a}, \triangle y = \frac{y_c^g - y_c^a}{d^a}, \triangle z = \frac{z_c^g - z_c^a}{d^a}, \\ &\triangle l = log(\frac{l^g}{l^a}), \triangle w = log(\frac{w^g}{w^a}), \triangle h = log(\frac{h^g}{h^a}), \\ &\triangle\theta = \theta^g - \theta^a \end{aligned} \tag{4.3}$$

where $d^{\mathrm{a}} = \sqrt{(l^{\mathrm{a}})^2 + (w^{\mathrm{a}})^2}$ is the diagonal of the base of the anchor box.

Total Loss Function

The total loss function is defined as:

$$L = \alpha \frac{1}{N_{\mathrm{pos}}} \sum_i L_{\mathrm{cls}}({p_{\mathrm{i}}}^{\mathrm{pos}}, 1) + \beta \frac{1}{N_{\mathrm{neg}}} \sum_j L_{\mathrm{cls}}({p_{\mathrm{j}}}^{\mathrm{neg}}, 0) + \frac{1}{N_{\mathrm{pos}}} \sum_i L_{\mathrm{reg}}(u_{\mathrm{i}}, {u_{\mathrm{i}}}^*) \tag{4.4}$$

where ${p_{\mathrm{i}}}^{\mathrm{pos}}$ and ${p_{\mathrm{j}}}^{\mathrm{neg}}$ mean the softmax output for positive anchor ${a_{\mathrm{i}}}^{\mathrm{pos}}$ and negative anchor ${a_{\mathrm{j}}}^{\mathrm{neg}}$ respectively, while $u_{\mathrm{i}} \in \mathbb{R}^7$ and ${u_{\mathrm{i}}}^* \in \mathbb{R}^7$ are the regression output and ground truth for positive anchor ${a_{\mathrm{i}}}^{\mathrm{pos}}$. L_{cls} stands for binary cross entropy loss and α, β are two positive constants to balance between the negative and positive classification loss. we have used SmoothL1 function denoted as L_{reg} for regression loss [Zhou 18].

4.3 Training Details

4.3.1 Datasets

The chosen model is trained using two different datasets, generated by Kitti Vision Benchmarking Suite [Geiger 12] and Waymo Open Dataset [Sun 20].

4.3.1.1 Kitti Dataset

The Kitti dataset was originally released by Geiger et al. (2012) to help researchers in designing advanced models for object detection and segmentation both in 2D and 3D. The dataset is recorded in and around Karlsruhe, a city in Germany. The below figure depicts the vehicle equipped with different sensors used for data recordings.

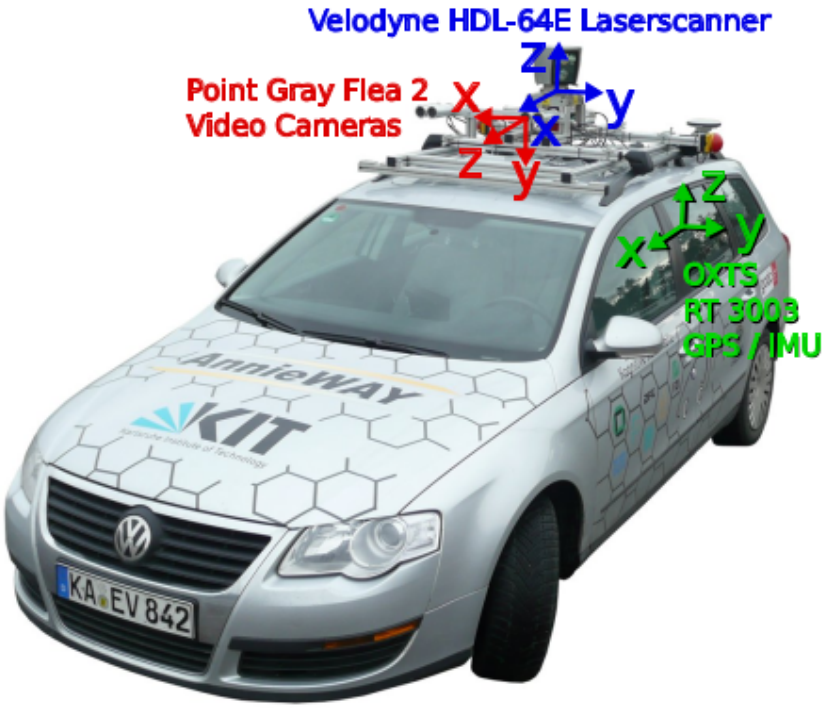


Figure 4.8: The vehicle used for recording Kitti dataset with sensors setup [Geiger 12].

Followings are the sensors used in Kitti:

- 1 Inertial Navigation System (GPS/IMU): OXTS RT 3003
- 1 Laser scanner: Velodyne LiDAR HDL-64E
- 2 Grayscale cameras, 1.4 Megapixels: Point Grey Flea 2(FL2-14S3M-C)
- 2 Color cameras, 1.4 Megapixels: Point Grey Flea 2(FL2-14S3C-C)
- 4 Varifocal lenses, 4-8 mm: Edmund Optics NT59-917

The point cloud of around 100k is captured per scan by spinning the LiDAR at 10 frames per second. The resolution of an image is 1382x512 pixels. The Kitti has provided 3D bounding boxes for cars, trams, vans, pedestrians, and cyclists. The annotation is done manually based on camera information such that only objects in the camera plane are considered to avoid false positives.

Figure 4.9: An example of Kitti image frame[Geiger 12].

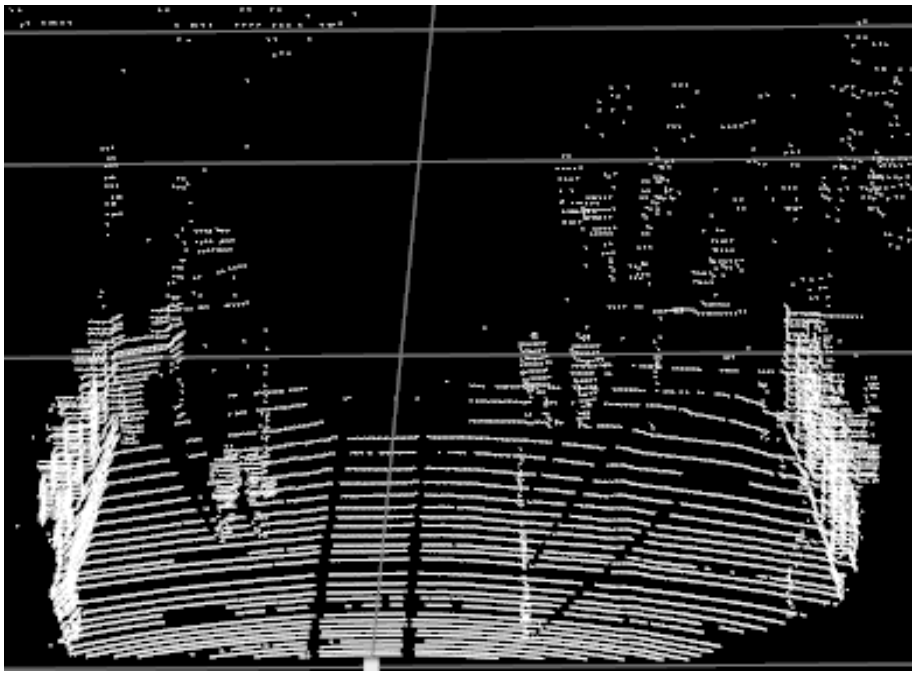


Figure 4.10: An example of Kitti LiDAR frame [Geiger 12].

The dataset consists of 7481 training frames(including point clouds and images), used for training and validation, and 7518 for testing without labels. The Kitti has considered three levels for detection evaluation: easy, moderate, and hard, based on the object size, occlusion, and truncation.

4.3.1.2 Waymo Dataset

The Waymo data set was released in 2020 by Waymo, a sister company of Google to tackle the scarcity of data and to provide more data to the research community in the field of perception and motion planning. Currently, it is one of the largest commercial datasets available. The dataset is recorded in and around the cities of the United States of America, namely San Francisco, Phoenix, and Mountain View.

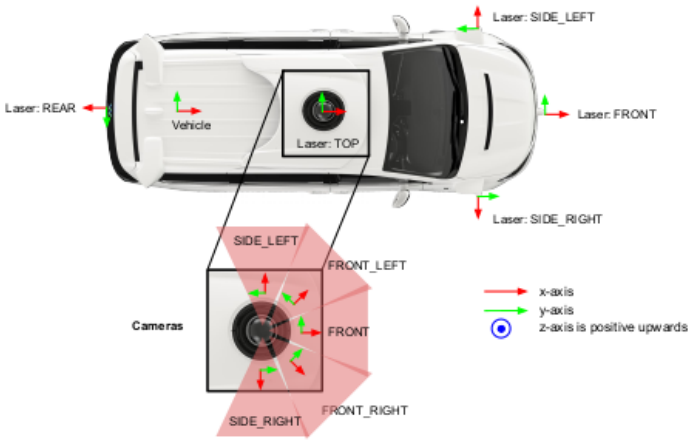


Figure 4.11: The vehicle used for recording Waymo dataset with sensors setup [Sun 20]

.

The Waymo has used four mid-range LiDAR, one long range LiDAR, five cameras with image resolution of 1920x1280. The number of points captured per scan from LiDAR is around 220k, which is almost twice the points in Kitti. Unlike Kitti, Waymo developed all of the sensors in-house, so no exact information is given about sensors specification.

In this work, we have used only 7525 frames(after removing the frames where there are no relevant objects for the task) out of 230k frames in which 5981, 771, and 771 are used for train, valid, and test, respectively. The Waymo has provided the 3D bounding boxes for vehicles(cars, trucks, vans, and buses), pedestrians, cyclists, and signs. Since the Waymo data set is converted into Kitti data format, evaluation of detection is done at the hard level.

Figure 4.12: Waymo dataset image frame example[Sun 20].

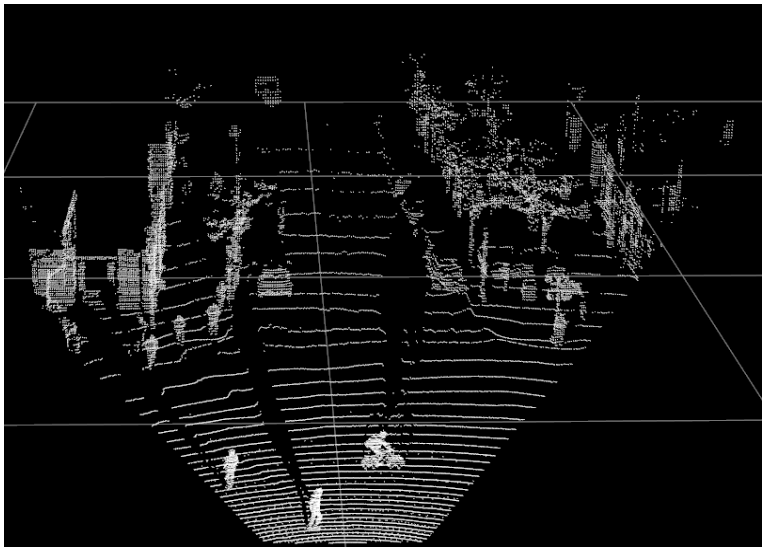


Figure 4.13: Waymo dataset LiDAR frame example [Sun 20].

Difficulties in Data-Sets

In most of the open-source data sets, only objects in the image plane are annotated and ground truths are in camera co-ordinate system. Therefore, LiDAR data should be projected onto the image plane first using projection and transformation matrices before using.

The difficulties in using Waymo data-set for 3D object detection with LiDAR:

- Finding the right conversion code to convert the data from Waymo format into Kitti format
- While converting the LiDAR point cloud data storage format from Waymo into Kitti .bin format, there will be lots of missing files of an image, point cloud, and ground-truth. Therefore, one has to examine the missing files to make sure that data conversion is correct
- Because of lots of up and downhill roads in Waymo data-set, taking the average of projection and transformation matrices of all frames to convert the ground-truths from camera coordinate system into LiDAR co-ordinate system is not possible, like in Kitti.
- Because of the above point, It needs lots of changes in voxelnet model code-base(voxelnet code is written specifically for the Kitti dataset) to make sure ground truths are correct.

Ground Truth

The Ground truths are removed based on the distances. For example, for distance 32 m, only ground truths within 32 m distance are kept for training and evaluation. Similarly, for distance 48 m.

4.3.2 Network Details

The experimentation is structured based on the LiDAR specifications of individual datasets Kitti and Waymo. In this work, the experimentation was conducted for different classes while considering a single class at a time as well as two classes at a time, mainly pedestrian and cyclist for a distance of 32 and 48 m.

4.3.2.1 Detection in Kitti

Pedestrian Detection:

For a distance of 32 and 48 m, point cloud range(scene size) of [-3,1] x [-20,20] x [0,32] and [-3,1] x [-20,20] x [0,48] m are considered along Z, Y, X respectively. By projecting the point clouds onto an image plane using projection and transformation matrices, point clouds outside the image plane are removed. we adopted a voxel size of $v_D = 0.4$, $v_H = 0.2$, $v_W = 0.2$ m with maximum number of points for each non-empty voxel T = 45, which leads to $D' = 10$, $H' = 200$, $W' = 160$ and $D' = 10$, $H' = 200$, $W' = 240$ for a distance of 32 and 48 m respectively.

In the model, two VFE layers VFE-1(7,32) and VFE-2(32,128) are used and finally VFE-2 output is mapped into $\mathbb{R}^{128}$ using FCN. Hence a sparse tensor of shape 128 x 10 x 200 x

160 and 128 x 10 x 200 x 240 are yielded by feature learning network for a distance of 32 and 48 m respectively. In middle convolution layer, three 3D convolution layers are used sequentially as Conv3D(128, 64, 3, (1,1,1), (1,1,1)), Conv3D(64, 64, 3, (1,1,1), (0,1,1)) and Conv3D(64, 64, 3, (2,1,1), (1,1,1)), which generates a 4D tensor of size 64 x 2 x 200 x 160 and 64 x 2 x 200 x 240 for 32 and 48 m distance respectively. The feature map of size 128 x 200 x 160 (distance 32 m) and 128 x 200 x 240 (distance 48 m) is input to RPN, where the dimensions of 3D tensor corresponds to channel, width and height. The output of the RPN is regression and probability score of pedestrian as shown is section 4.7. The values of the hyper-parameters in equation 4.2 are $\alpha = 1.5$ and $\beta = 1.0$.

Pedestrian and Cyclist Detection:

When the pedestrian and cyclist are considered at the same time, certain modifications to the output of the RPN are made, as shown in section 4.7 as well as the final loss function for both classification and regression. The rest of the model structure is kept similar to pedestrian detection for both 32 and 48 m distance.

Final loss function is defined as:

$$\begin{aligned} L_{cls} &= \gamma * L_{classpedestrian} + \omega * L_{classcyclist} \\ L_{reg} &= \gamma * L_{regressionpedestrian} + \omega * L_{regressioncyclist} \\ L &= L_{cls} + L_{reg} \end{aligned} \tag{4.5}$$

Where L_{cls} stands for overall classification loss of both pedestrian and cyclist, L_{cls} stands for overall regression loss of both pedestrian and cyclist and γ, ω are positive constants balancing the relative importance which are set to 1 and 1.3 respectively. $L_{\text{classpedestrian}}$, $L_{\text{classcyclist}}$, $L_{\text{regressionpedestrian}}$ and $L_{\text{regressioncyclist}}$ stands for binary cross entropy loss and regression loss for pedestrian and cyclist respectively.

4.3.2.2 Detection in Waymo

For Waymo, similar architecture is used as Kitti with a small changes in Z axis of scene size as explained in section 4.3.2.1, where value of Z is hyperparameter depending upon data set. Here, we considered point cloud range of [-2,4] x [-20,20] x [0,32] and [-2,4] x [-20,20] x [0,48] for 32 and 48 m distance respectively.

Since the scene size is increased, model complexity also increased, which means channel size of the input feature map to the RPN as explained in the section 4.3.2.1 To reduce the model complexity, 1x1 convolutional layer is added to reduce the channel size of feature map.

4.3.3 Framework

The DL framework, Tensorflow is used for the implementation of voxelnet. The training and testing were run on an Nvidia GeForce GTX 2080 Ti on a Neptun server with CUDA 11.1. For the analysis and visualization of LiDAR point cloud and predicted 3D bounding boxes, we have used mayavi[Enthought 21]. The majority of the implementation is based on [Zhou 18].

4.3.4 Anchors

Anchor is a priori for the size of the detected object. Anchor with different scales and aspects ratio makes the network more stable and enable faster convergence for the model. For efficiency reasons, we have used two anchors for each class.

Kitti

- Pedestrian:
  1. $l_a = 0.8m$, $w_a = 0.6m$, $h_a = 1.73m$ and $\theta_a = 0°$
  2. $l_a = 0.8m$, $w_a = 0.6m$, $h_a = 1.73m$ and $\theta_a = 90°$
- Cyclist:
  1. $l_a = 1.76m$, $w_a = 0.6m$, $h_a = 1.73m$ and $\theta_a = 0°$
  2. $l_a = 1.76m$, $w_a = 0.6m$, $h_a = 1.73m$ and $\theta_a = 90°$

Waymo

- Pedestrian:
  1. $l_a = 0.9m$, $w_a = 0.86m$, $h_a = 1.73m$ and $\theta_a = 0°$
  2. $l_a = 0.9m$, $w_a = 0.86m$, $h_a = 1.73m$ and $\theta_a = 90°$
- Cyclist:
  1. $l_a = 1.76m$, $w_a = 0.86m$, $h_a = 1.73m$ and $\theta_a = 0°$
  2. $l_a = 1.76m$, $w_a = 0.86m$, $h_a = 1.73m$ and $\theta_a = 90°$

4.4 Evaluation Metrics

To evaluate the performance of any machine learning/deep learning model over validation/test set, there will be a metric depending upon the application and use case. Similarly, Mean Average Precision aka, the mAP is used to evaluate the performance of the 3D object detection. Before going to the calculation part of mAP, let's discuss the precision and recall curve.

4.4.1 Precision and Recall

Consider an object classification task, where a deep learning model has to classify between pedestrian and cyclist. Given 10 pedestrian objects in a frame, a model classifies one of ten pedestrians correctly with no miss classification, then the model has high precision, but recall of the model is very less(one of 10). The recall measures the ability of a model to classify all the interesting objects in a data set.

There exists a trade-off between precision and recall. For example, when the false positives(classifying cyclists as a pedestrian, even though an object is a cyclist) are avoided more than false negative(not classifying pedestrian as a pedestrian), then the model should produce the high precision prediction by lowering the recall.

The precision is defined as:

$$precision = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \tag{4.6}$$

The recall is defined as:

$$recall = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \tag{4.7}$$

4.4.2 Average Precision

Average precision measures whether your model can accurately identify all positive instances without mistakenly labeling too many negative ones as positive. As a result, when the model can handle positives appropriately, average accuracy is high. The area under a curve that reflects the trade-off between precision and recall at different chosen thresholds is used to determine average precision:

An ideal model with precision 1 and recall 1 would have zero mistakes in classifying the objects. In practice, it is very difficult to achieve.

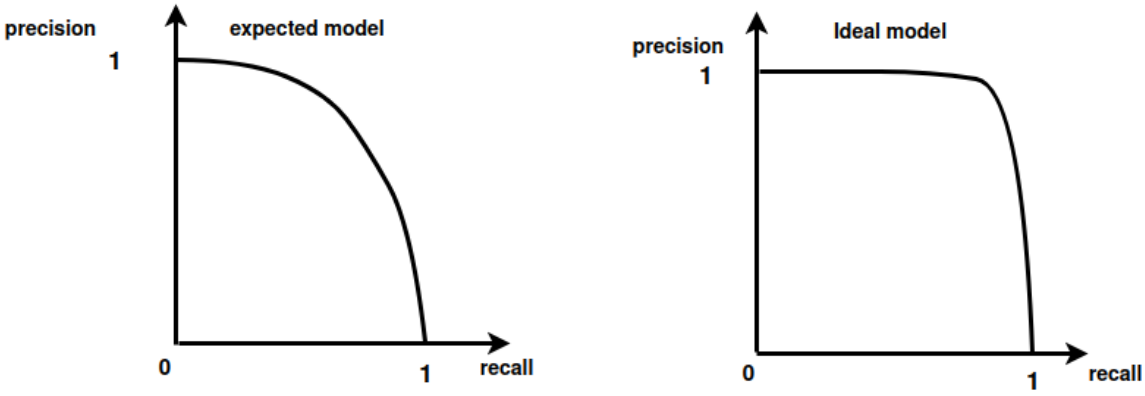


Figure 4.14: The average precision curve.

Average precision (AP) is a numerical summary for the shape of the precision-recall curve. AP is defined as mean precision at N discrete values of recall, whose values lie between 0 and 1.

$$AP = \frac{1}{N} \sum_{\mathrm{N}=1}^{N} \max p(r_i) \tag{4.8}$$

Where $p(r_i)$ is the measured precision at recall r_i.

the mAP is the average of AP over a certain number of epochs.

5. Class Imbalance

Imbalance is an extensively studied topic in deep learning and computer vision. Since the focus of the thesis is on object detection, we will restraint the problem to object detection in point clouds. There are four main types of imbalance, namely: class imbalance, spatial imbalance, scale imbalance, and objective imbalance [Oksuz 20].

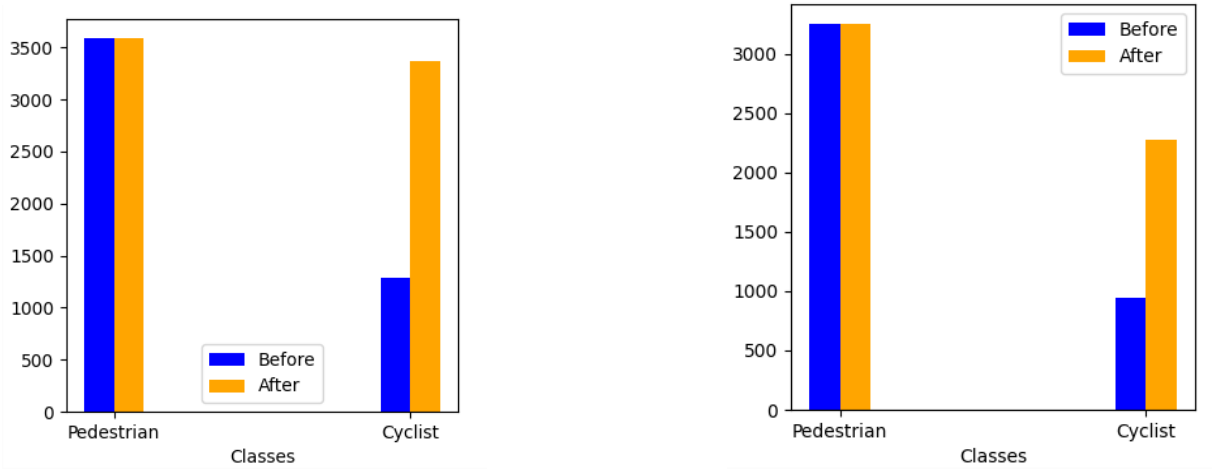
Class balance exists when the number of two object instances are not balanced properly, either foreground to background or foreground to foreground. Occurrence of scale imbalance when the number of reflected points from the objects differ. For example, the number of reflected points from the pedestrian at a distance of 48 m is very less compared to the pedestrian at a distance of 32 m. Spatial imbalance indicates a "set of factors related to spatial properties of the bounding boxes such as regression penalty, location and IoU"[Oksuz 20]. Lastly, minimization of multiple loss functions(e.g. two cross-entropy loss functions in multi-label object detection) leads to object imbalance problem[Oksuz 20].

Zhu et al. [Zhu 19] proposed consideration of multi-category joint detection drives to multi-task learning problems in a point cloud. In order to achieve good performance for the multi-category joint object detection, at least the number of examples related to various classes should be almost equal, whereas this is not the case with Kitti and Waymo data set. So we will discuss the imbalance problem in each data set and solution in section 5.1.

According to Phan et al. [Phan 20], the wide adaption of vanilla cross-entropy loss function for object classification easily neglects the occurrence frequency of objects during training and the outcome of this approach would be lower accuracies for under-represented classes. This problem is overcome by using the weighted cross-entropy loss function which is discussed briefly in section 5.1.

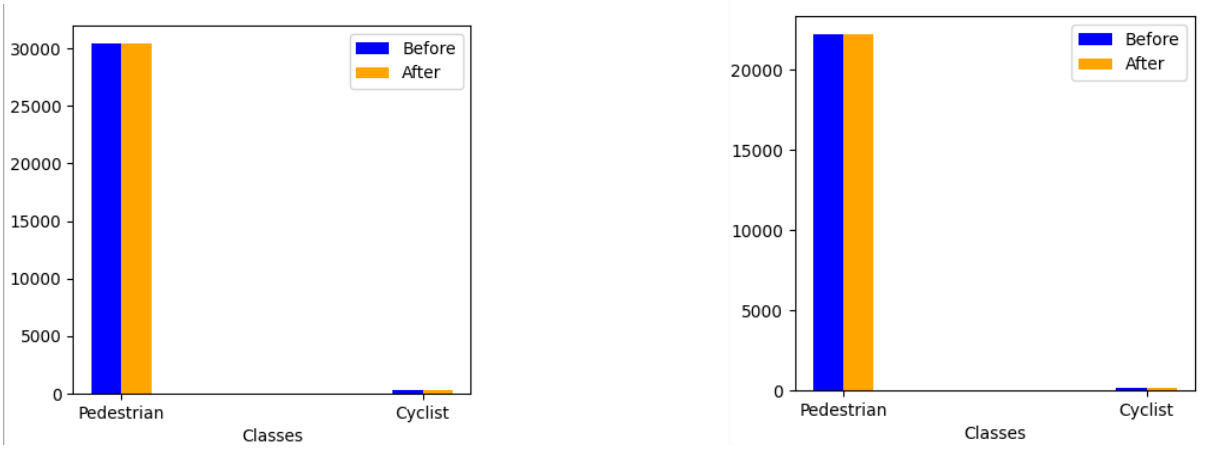
5.1 Balanced Sampling

To mitigate the class imbalance problem, we followed the method called Dataset sampling(DS Sampling), proposed by Zhu et al. We slightly modified the DS Sampling, instead of duplicating the fraction of specific samples, we tried to minimize the distribution error. At the end, which brings the difference in the distribution of pedestrian and cyclist in Kitti data set to 30 points for a distance of 32 m and almost 0 for a distance of 48 m. The



(a) Distance48 **(b)** Distance32

Figure 5.1: Class imbalance in the Kitti Dataset.



(a) Distance48 **(b)** Distance32

Figure 5.2: Class imbalance in the Waymo Dataset.

object's instances distribution after the DS sampling can be seen in the above figures in orange columns.

Whereas, in Waymo, the distribution of the object's instance is severely skewed, doing DS sampling requires large memory to store the duplicates of samples, mainly cyclist frames, and requires more time to train as well. In considering the disadvantages of DS sampling on Waymo, we recommend getting more instances of cyclists and do the DS sampling.

Along with DS Sampling, we have applied data augmentation like global scaling and rotation to the whole point cloud and all ground truth boxes. The uniform distribution between [0.95,1.05] is used for global scaling and rotation angle between [-pi/4,pi/4] is adopted for rotation as stated in [Zhu 19].

To conclude, the impact of DS sampling on Kitti is discussed in section 6.2.2 and we have not done any kind of DS sampling for single-class object detection.

6. Experiments and Results

In this chapter, the results of the experiments on voxelnet on both Kitti and Waymo datasets are shown.

6.1 Kitti Evaluation Protocol

Since the evaluation of the model on Kitti server is limited only for researchers, we have split the training data into train, valid, and test, each has 5981, 750, and 750 frames respectively. According to Kitti evaluation protocol, the performance of the model would be measured in terms of average precision(AP) with an IoU threshold of 0.5 for both pedestrians and cyclists.

6.2 Evaluation on Kitti Dataset

The model is evaluated across all three difficulty levels in Kitti, The levels are: easy, moderate, and hard. Where occlusion and truncation level for easy is between 100% and 75%, for moderate is between 75% and 50%, for hard 50% and 25%. Then the results are compared with some SOTA models in 3D object detection.

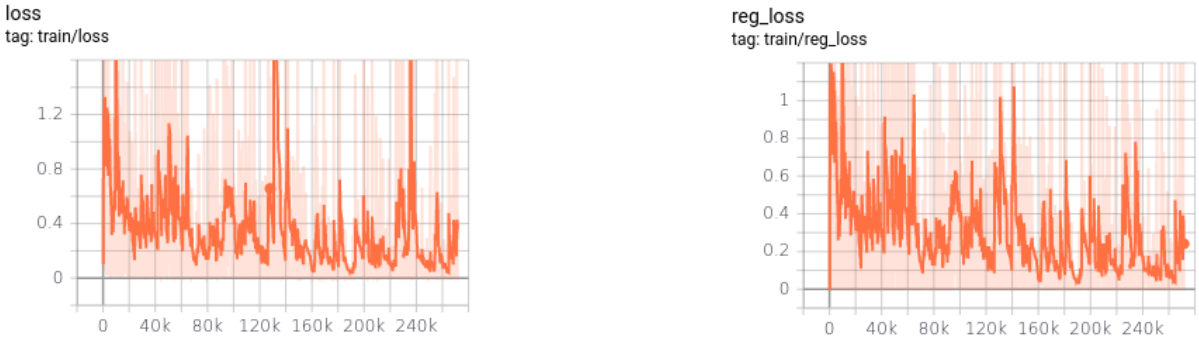
Two evaluation metrics are computed: localization average precision (AP_{loc}) and 3D bounding box detection average precision (AP_{3D}). Bird's eye view(2D ground plane) is used for measuring the AP_{loc} and AP_{3D} is evaluated on world space(3D space).

The presented results are mAP for 160 epochs.

6.2.1 Loss Graphs

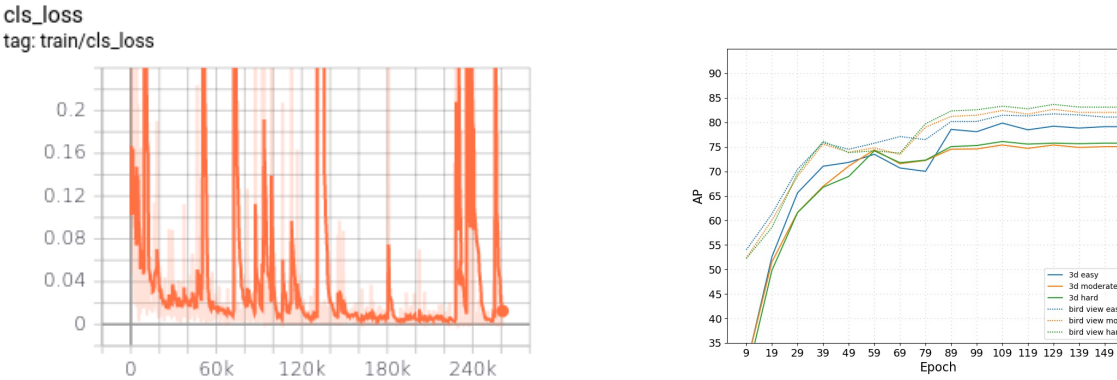
Single Class

The overall classification and regression loss is the combined classification and regression loss.



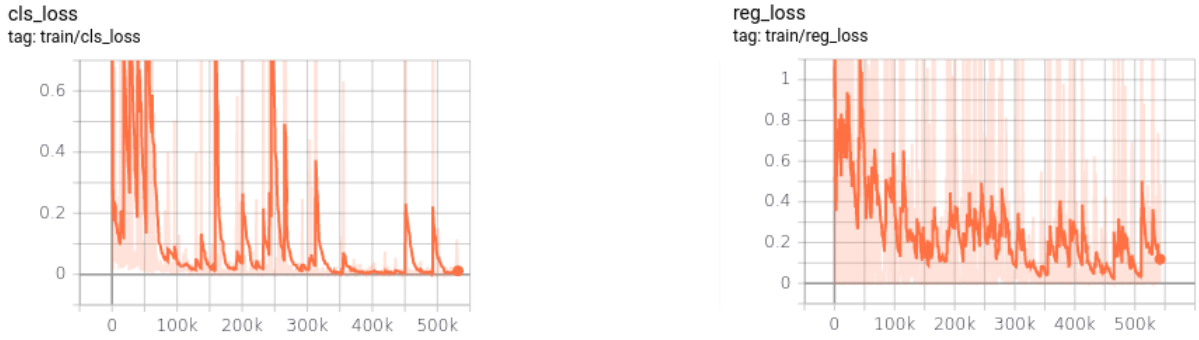
(a) Overall loss **(b)** Regression loss

Figure 6.1: The graphs show (a) the overall loss, (b) regression loss for single class on Kitti for a distance of 48 m.



(a) Overall classification loss **(b)** Average precision curve

Figure 6.2: The graphs show (a) the overall classification loss, (b) average precision curve for single class on Kitti for a distance of 48 m.

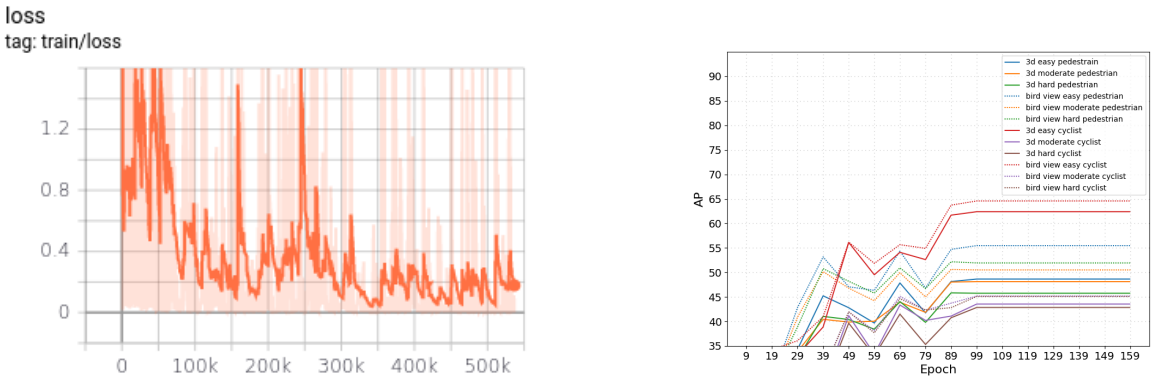


(a) Overall classification loss **(b)** Overall regression loss

Figure 6.3: The graphs show (a) the overall classification loss, (b) overall regression loss for multi-label on Kitti.

Multi Label

The overall loss is the combined loss of regression and classification loss of both pedestrian and cyclist.



(a) Overall loss **(b)** Average precision curve

Figure 6.4: The graphs show (a) the overall loss, (b) average precision curve for a distance of 48 m for multi-label on Kitti.

6.2.2 Results

Single Class

This section describes the performance of the model when considering only pedestrians.

Validation Set

[Zhou 18] has considered total of ≈ 1500 frames for validation. Since Kitti has no access to the server for master students, approximately 750 frames are used for validation in order to use data for testing as well.

Method	Speed(Hz)	Modality	Pedestrian			
			View	Easy	Moderate	Hard
YOLO3D (BEV)	40	LiDAR	3D	N/A	N/A	N/A
			BEV	N/A	N/A	N/A
MV3D (BV+FV+RGB)	1	LiDAR+ Mono	3D	N/A	N/A	N/A
			BEV	N/A	N/A	N/A
Voxelnet (Vanilla)	3.3	LiDAR	3D	57.86	53.42	48.87
			BEV	65.95	61.05	56.98
Voxelnet (ours)	3.3	LiDAR	3D	**60.80**	**59.02**	**56.22**
			BEV	**68.48**	**65.05**	**63.83**

Table 6.1: Evaluation on the Kitti validation dataset for 48 m distance for single class.

YOLO3D and MV3D were tested on Titan X GPU, both versions of voxelnet is tested on Nvidia GeForce GTX 1080 Ti.

As we can see in the Table 6.1, the performance of voxelnet(ours) is more than vanilla voxelnet in both 3D and BEV in all difficulty levels, which is due to the change in data augmentation.

Since the most of the models are not evaluated for a distance of 32 m, only the results of voxelnet(ours) is shown.

Method	Time(s)	Modality	Pedestrian			
			View	Easy	Moderate	Hard
Voxelnet(ours)	3.3	LiDAR	3D	**59.07**	**58.78**	**56.26**
			BEV	**64.21**	**63.78**	**63.04**

Table 6.2: Evaluation on the Kitti validation dataset for 32 m distance for single class.

The results of the model for a distance of 32 m are not good as compared to the results of 48 m distance in both 3D and BEV. The main reason might be, the number of instances available to the network, where the number of pedestrian instances is more in distance 48 m as compared to the distance of 32 m.

Experiment 1

The noise was added in the input data(for each frame) to the network, the results of the prediction were not affected by the addition of noise. The Figure 6.5 illustrates the addition of noise into the input data.

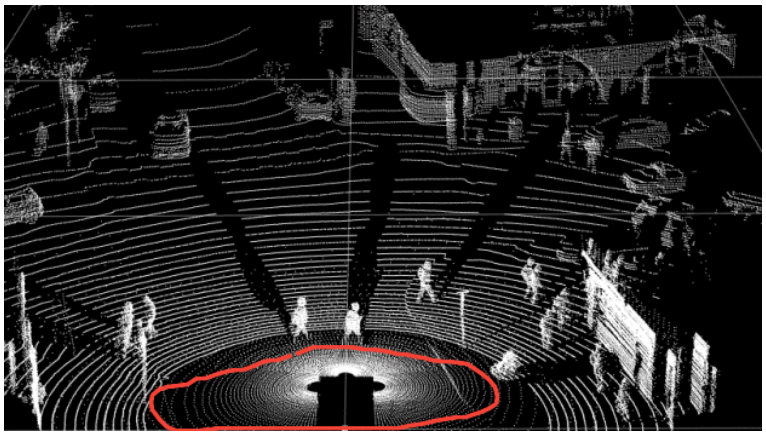


Figure 6.5: The visual representation of addition of noise into the input data [Sun 20].

Testing Set

The results of the model on the testing set are not compared with any of the models. Since most of the models presented their results in the paper, after the evaluation on testing data set available on Kitti remote server.

Method	Modality	Pedestrian			
		View	Easy	Moderate	Hard
Voxelnet(ours)	LiDAR	3D	**56.31**	**53.20**	**52.09**
		BEV	**61.90**	**61.50**	**59.38**

Table 6.3: Evaluation on the Kitti testing dataset for 48 m distance for single class.

Method	Modality	Pedestrian			
		View	Easy	Moderate	Hard
Voxelnet(ours)	LiDAR	3D	**5.31**	**58.59**	**59.62**
		BEV	**65.43**	**62.64**	**60.60**

Table 6.4: Evaluation on the Kitti testing dataset for 32 m distance for single class.

Multi Label

As discussed already in 4.1.3, the single-class 3D object detector is extended to multi-label 3D object detector as shown in 4.7.

From our literature review and research, we have found that only limited networks considered pedestrian and cyclist at the same time in their work.

Experiment 2

To work on multi-label network, number of instances of classes should be at least uniformly distributed. But, in both the data-sets(Kitti and Waymo), it was not. Initially, the multi-label model is trained without balancing the instances of classes. But, the results are not good. So, DS sampling approach as explained in 5.1 is used to balance the instances of classes. The effect of DS sampling is shown in the following sections.

Validation Set

The vanilla voxelnet model did not consider both classes, pedestrian and cyclist at the same time while evaluating the validation set, and also most of the models based on different feature extractors(pointnet++ and point pillar) were provided the results only for car class on the validation set, except F-Pointnet [Qi 18].

Method	Speed(Hz)	Modality	Cyclist			
			View	Easy	Moderate	Hard
Voxelnet(ours)	3.3	LiDAR	3D	**52.03**	**36.77**	**35.74**
			BEV	**53.54**	**38.95**	**38.71**
F-pointnet	5.9	LiDAR+RGB	3D	**77.15**	**56.49**	**53.37**
			BEV	**81.82**	**60.03**	**56.32**

Table 6.5: Evaluation on the Kitti validation dataset for 48 m distance for cyclist.

Method	Modality	Pedestrian			
		View	Easy	Moderate	Hard
Voxelnet(ours)	LiDAR	3D	**42.73**	**41.54**	**40.34**
		BEV	**49.33**	**47.92**	**45.10**
F-pointnet	LiDAR+RGB	3D	**70.00**	**61.32**	**53.59**
		BEV	**72.38**	**66.39**	**59.57**

Table 6.6: Evaluation on the Kitti validation dataset for 48 m distance for pedestrian.

Method	Modality	Pedestrian				Cyclist		
		View	Easy	Moderate	Hard	Easy	Moderate	Hard
Voxelnet (ours)	LiDAR	3D	**48.52**	**47.14**	**47.34**	**56.20**	**50.86**	**50.53**
		BEV	**55.06**	**53.02**	**52.45**	**58.40**	**54.12**	**52.83**

Table 6.7: Evaluation on the Kitti validation dataset for 32 m distance.

Testing Set

The voxelnet vanilla model did not consider both classes, pedestrian and cyclist at the same time while doing evaluation, so the voxelnet(ours) is not compared with any of the models.

Method	Modality	Pedestrian				Cyclist		
		View	Easy	Moderate	Hard	Easy	Moderate	Hard
Voxelnet (ours)	LiDAR	3D	**35.92**	**36.74**	**36.78**	**37.07**	**28.92**	**28.64**
		BEV	**43.91**	**44.26**	**44.89**	**40.40**	**29.93**	**30.28**

Table 6.8: Evaluation on the Kitti testing dataset for 48 m distance.

Method	Modality	Pedestrian				Cyclist		
		View	Easy	Moderate	Hard	Easy	Moderate	Hard
Voxelnet (ours)	LiDAR	3D	**43.74**	**42.72**	**42.44**	**43.12**	**42.30**	**42.16**
		BEV	**52.50**	**51.32**	**48.56**	**45.90**	**45.62**	**45.43**

Table 6.9: Evaluation on the Kitti testing dataset for 32 m distance.

6.3 Evaluation on Waymo dataset

The results of voxelnet(ours) are evaluated on the Waymo dataset using the same evaluation criteria defined by Kitti. One thing to note here, Waymo has defined its level of difficulty based on the number of points visible to the annotators and it is up-to-the annotators to define the level of difficulty. Hence, there is no specific difficulty level in Waymo like in Kitti. Therefore, the hard level of difficulty is used for the evaluation. Approximately, 750 frames are taken for both validation and testing respectively.

Experiment 3

Initially, Waymo dataset consists of large number of up and down hill LiDAR scans as shown in the Figure 6.6. When the model is trained on dataset with up and down hill LiDAR scans, results were not good. Hence, scene height adoption experiment was conducted. But, it is still not clear that whether the up and down hill roads had any impact on the results. Therefore, the relationship between non-flat road surface and height of the scene should be properly analyzed.

Figure 6.6: The visual representation of down hill road in Waymo dataset [Sun 20].

Experiment 4

After the removal of up and down hill LiDAR scans, still results of the model were not good. Hence, experimentation on finding the optimal height of the scene was conducted. Since the results were not available currently, results of the model before the proper scene height adoption were not presented here.

But, results are shown in terms of visualization in the Figure 6.7

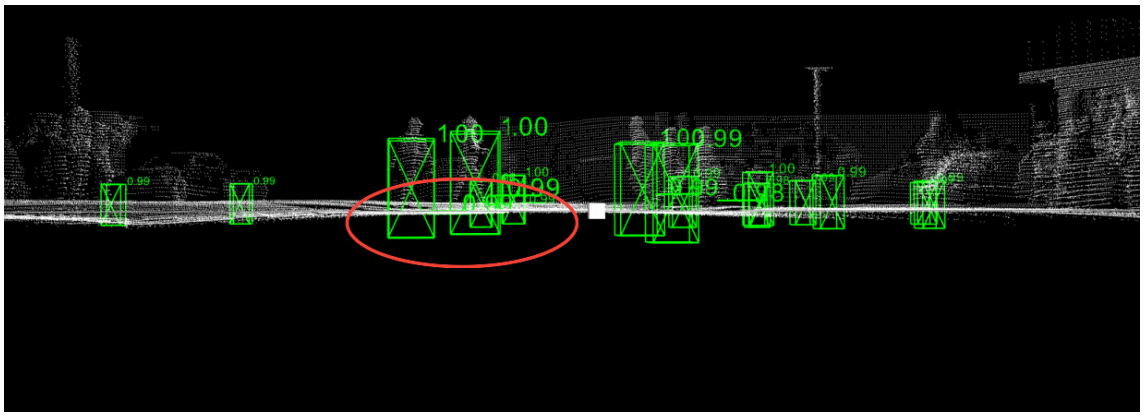


Figure 6.7: The visualization of model prediction on Waymo dataset before the scene height adoption [Sun 20]. The predicted bounding boxes are not perfectly aligned with the objects.

Based on the above experimentation, an approach was proposed to find the optimal height of the scene in the section 6.6.

6.3.1 Results

Single class

In this section, the performance of the model is presented on Waymo for the pedestrian object. The results of the model on Waymo data-set is not compared with any of the models. The reason is the level of difficulty in the data-set, where there are ground truths in the frame, even the number of reflected points from the pedestrians are less than 5 and sometimes it goes to 1 as well.

Validation set

The vanilla voxelnet model is not trained and evaluated on Waymo dataset.

Method	Modality	Pedestrian	
		View	Hard
Voxelnet(ours)	LiDAR	3D	**40.35**
		BEV	**46.44**

Table 6.10: Evaluation on the Waymo validation set for 48 m distance.

Method	Modality	Pedestrian	
		View	Hard
Voxelnet(ours)	LiDAR	3D	**38.38**
		BEV	**42.04**

Table 6.11: Evaluation on the Waymo validation set for 32 m distance.

Testing set

Method	Modality	Pedestrian	
		View	Hard
Voxelnet(ours)	LiDAR	3D	**30.85**
		BEV	**37.12**

Table 6.12: Evaluation on the Waymo testing set for 48 m distance.

Method	Modality	Pedestrian	
		View	Hard
Voxelnet(ours)	LiDAR	3D	**30.12**
		BEV	**34.69**

Table 6.13: Evaluation on the Waymo testing set for 32 m distance.

Multi Label

We assume that it is inappropriate to evaluate the multi-label model on Waymo dataset, where the distribution of a number of instances of pedestrians and cyclists are not properly distributed and also for the reason of ground truth difficulty level in Waymo dataset. However, the results of the model on the testing set are presented.

Testing Set

Method	Modality	Pedestrian		Cyclist
		View	Hard	Hard
Voxelnet(ours)	LiDAR	3D	**14.22**	**6.80**
		BEV	**16.28**	**10.29**

Table 6.14: Evaluation on the Waymo testing dataset for 48 m distance.

Method	Modality	Pedestrian		Cyclist
		View	Hard	Hard
Voxelnet(ours)	LiDAR	3D	**27.01**	**12.31**
		BEV	**31.72**	**21.02**

Table 6.15: Evaluation on the Waymo testing dataset for 32 m distance.

6.4 Detection analysis

In this section, the performance of the model in terms of visualization is presented.

6.4.1 Kitti

Single Class

The pink color bounding box(BB) represents ground truth and sky-blue color BB represents pedestrian(predicted).

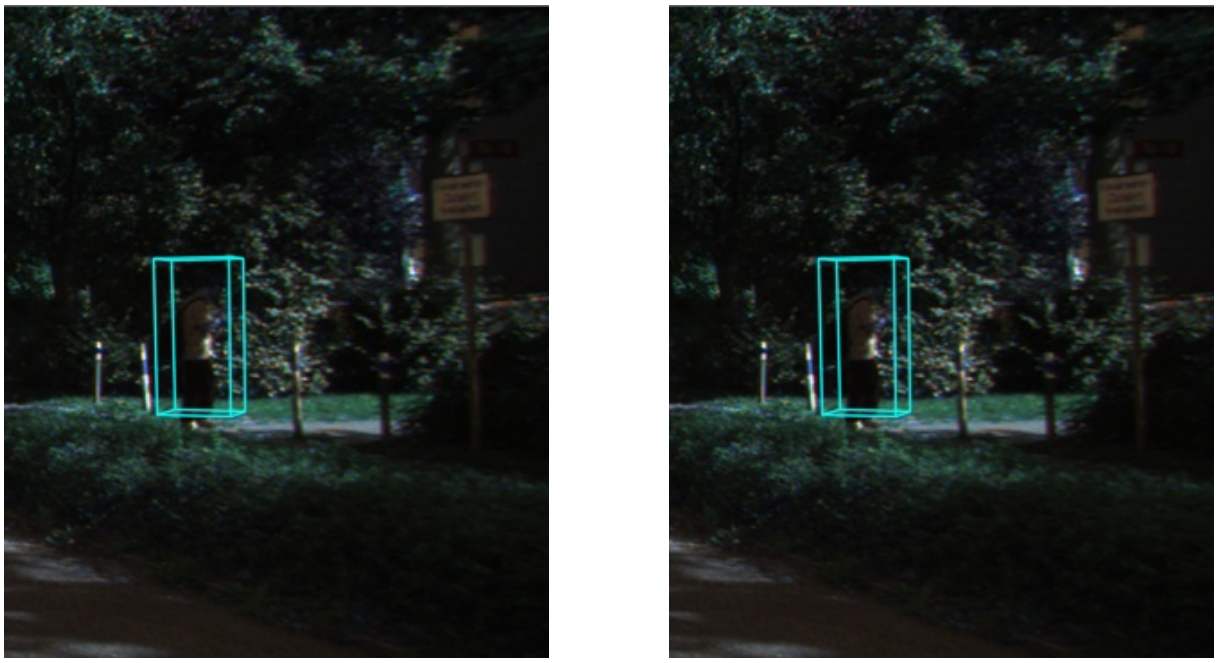
(a) Distance32 **(b)** Distance48

Figure 6.8: Signs are detected when the distance is 48 m in single-class voxelnet [Geiger 12].

As we can see in Figure 6.10, the tree trunk is detected as a pedestrian. This might be due to voxelization. Since the height of the scene(as explained in the section 4.1.1.1) is a hyperparameter in the voxelnet based model. Still, the effect of scene height should be analyzed thoroughly in near future.

(a) Distance32 **(b)** Distance48

Figure 6.9: Cyclist is detected as a pedestrian by both 32 m and 48 m distance models and pedestrians with no ground truths are detected as well in single-class voxelnet[Geiger 12].



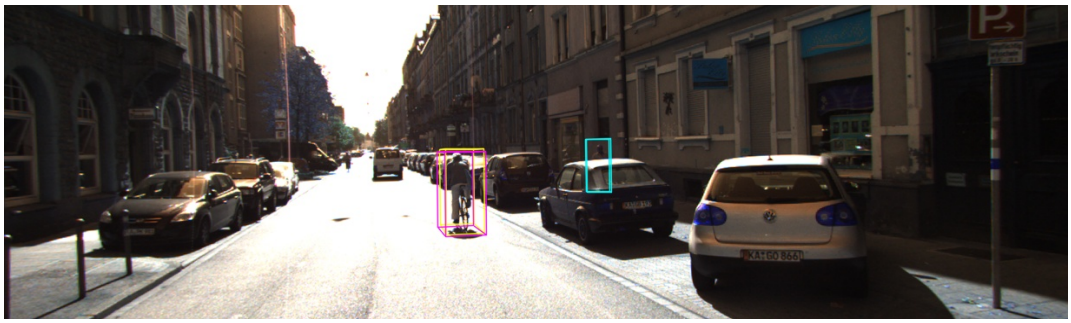
(a) Distance32 **(b)** Distance48

Figure 6.10: Tree trunk is detected as a pedestrian by both 32 m and 48 m distance models in single-class voxelnet. [Geiger 12].

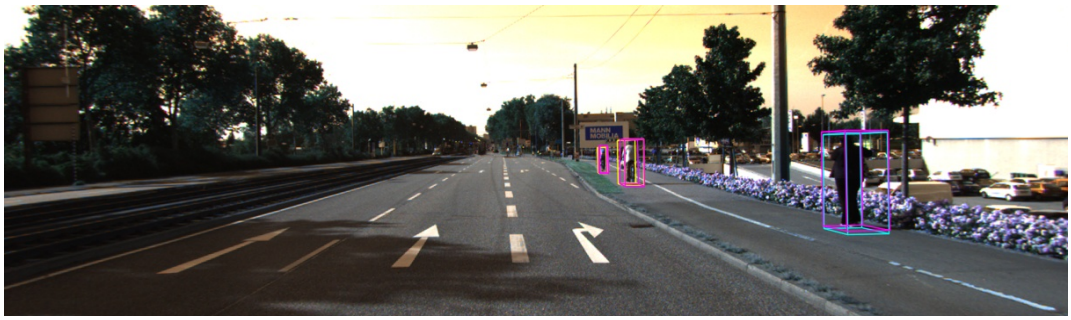
Multi Label

In multi-label, yellow color BB represents cyclist(predicted), pink color BB represents ground truth and sky-blue color BB represents pedestrian(predicted).

In a multi-label detection task, especially with LiDAR, as the distance of cyclist increases



(a) Distance32



(b) Distance48

Figure 6.11: Proper detection of cyclist and pedestrian in Kitti [Geiger 12].

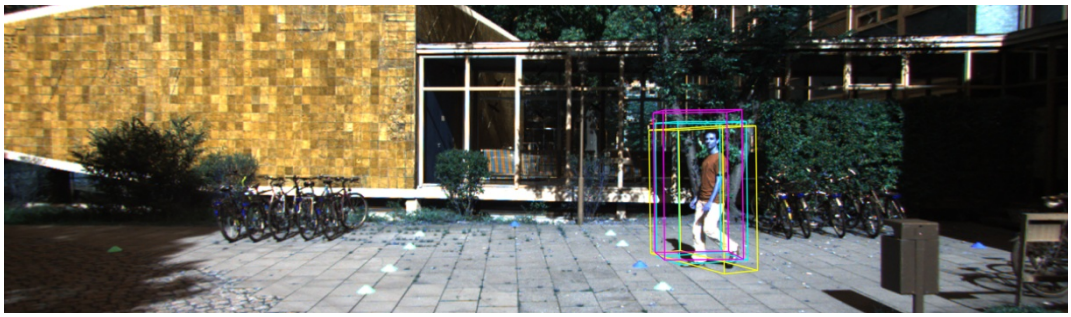
(a) Distance32

(b) Distance48

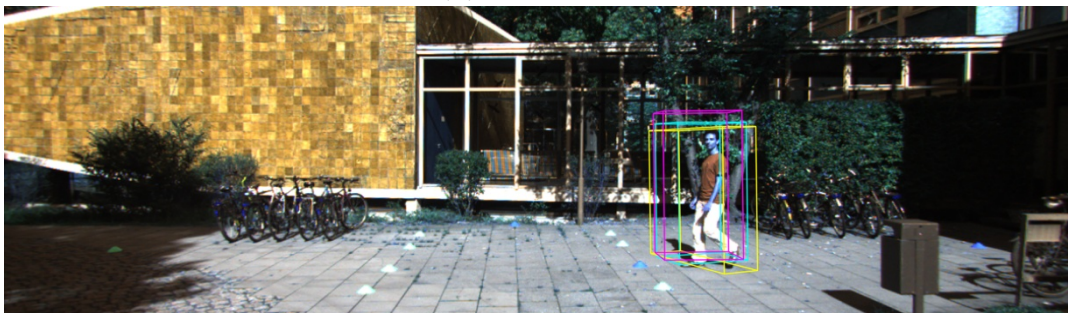
Figure 6.12: Proper detection of cyclist in complex situation in Kitti [Geiger 12].

from the position of LiDAR, the appearance or representation of the cyclist will be almost similar to the pedestrian. But, the model is capable of detecting far away objects as shown in the Figure 6.12. However, the model will detect the pedestrian as a cyclist even in the

near distance sometimes as shown in Figure 6.13.



(a) Distance32



(b) Distance48

Figure 6.13: Detection of pedestrian as cyclist even in nearest distance in Kitti [Geiger 12].

(a) Single-class

(b) Multi-label

(c) Random false positives. In few frames, random false positives are present in both single and multi-label model irrespective of distance in Kitti [Geiger 12].

The random false positives as shown in the Figure 6.14c might be due to the non-existence of points. The non-existence of points will occur, when only the projected point cloud onto the image plane is considered as shown in the Figure 4.13, results in a large number of empty voxels as shown in Figure 4.5.

Currently, most of the open-source data-sets provides the ground-truth of objects in image plane. But, for the pointnet++ or voxelnet or point pillar based 3D object detection models need the ground truth of the objects in LiDAR plane/frame. The following are the reasons:

- The detection models need the input data in the specified range in X, Y, Z axis as shown in the Figure 4.2, but the FOV in image plane will not cover the specified range in Y axis. So, there will be unoccupied space in the input data.

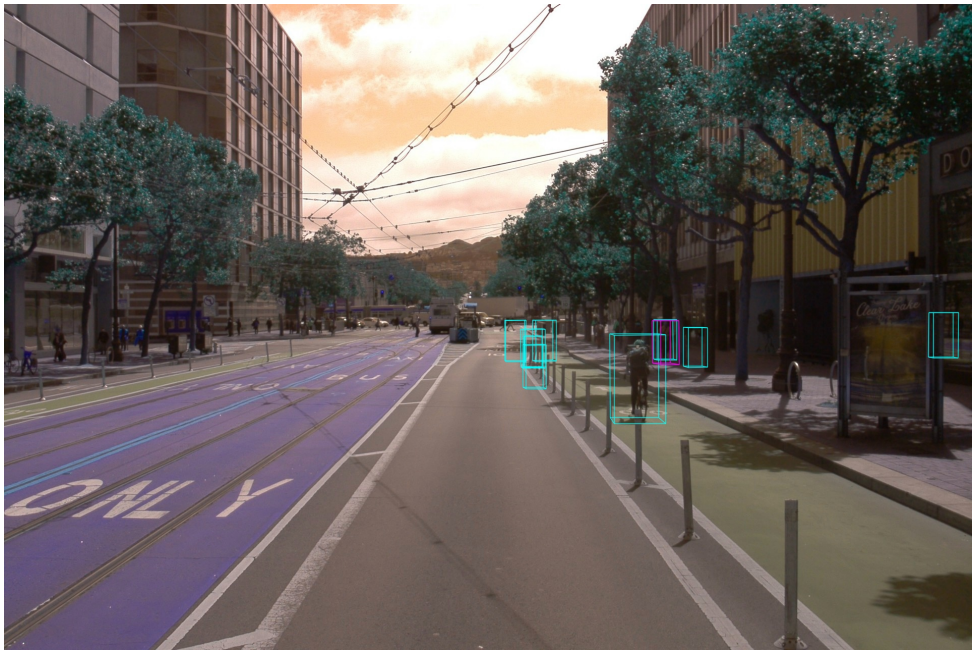
Therefore, voxelnet based models still need to be analyzed for the ground truths in LiDAR frame/plane.

6.4.2 Waymo

Since the distribution of the number of instances of pedestrians and cyclists is severely skewed in Waymo, the detection analysis is done only for single class.

Figure 6.15: Proper detection of pedestrian on Waymo [Sun 20].

(a) Distance32



(b) Distance48

Figure 6.16: A large number of false positives in every frame both in distance 32 m and 48 m distance as a result of high level of difficulty in ground truth in Waymo [Sun 20].

6.5 Evaluation on RR Lab Dataset

Due to the unavailability of ground truth in RR Lab dataset[Jan 20], the performance of the model is evaluated quantitatively.

As stated in section 6.6, the height of the scene is a hyperparameter in voxelnet based model, which makes the model not to perform well on the new data set. Nonetheless, Figure 6.17 shows the detection of pedestrians on RR Lab data set, where model trained on Kitti dataset. Sometimes, there will no detection as well.

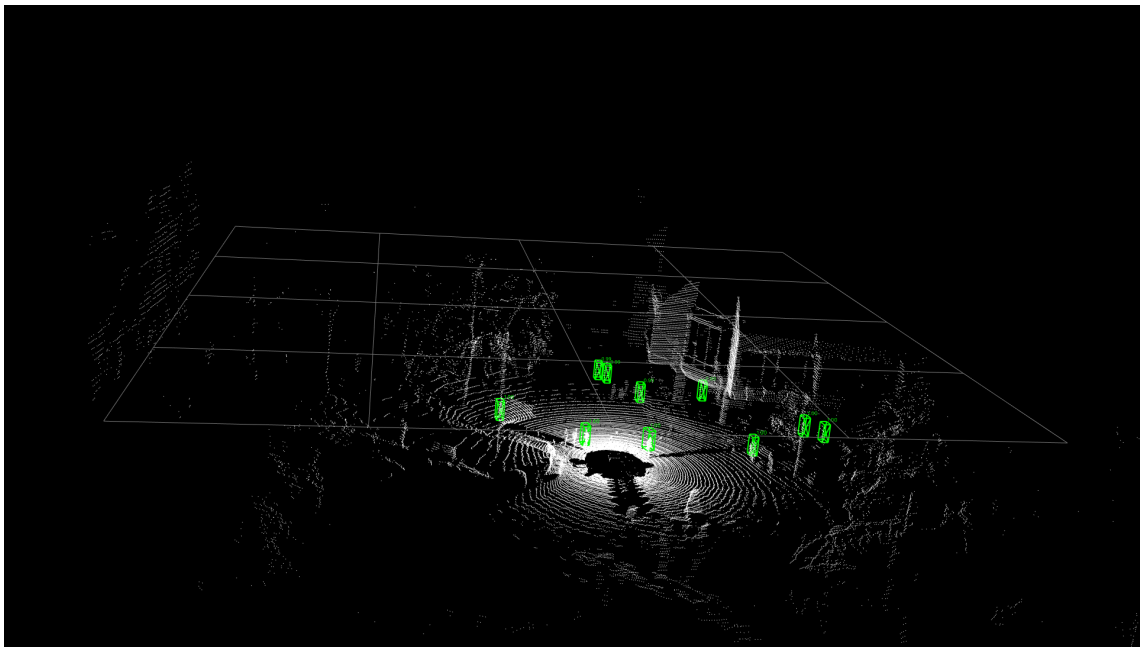


Figure 6.17: The detection of pedestrians on RR Lab dataset [Jan 20].

6.6 Experimentation with Hyper-parameters

Fine-tuning the model is an important skill in achieving good performance in deep learning. In the voxelnet based model, the height of the scene size 4.1.1.1 is a hyper-parameter apart from learning rate and batch size. Unlike learning rate and batch size, we can not choose the height randomly for different data sets. Hence, we have to come up with an approach, where we can adopt the scene height based on the position of the LiDAR on the vehicle.

In the Kitti dataset, the position of the LiDAR is 1.73 m from the ground, accordingly, we have chosen a scene height of 4 m from -3 m to 1 m. In the Waymo dataset, the position of the LiDAR is near to the ground. Thus we have selected a scene height of 6 m from -2 m to 4 m. From the above observation, we can say that as the position of the LiDAR goes up from the ground level, we need to choose more negative points in Z-axis(scene height). Additionally, we could have chosen a scene height of 5 m for the Waymo dataset, but the division of 5 by 0.4 (which is voxel size as explained in section 4.1.1.1) gives a float point number. Nevertheless, the value of the tensor should be int always.

The chosen value of hyper-parameters γ and ω in Kitti are 1 and 1, 1 and 1.3 for 32 m and 48 m distance respectively.

6.7 Transfer Learning

In the voxelnet based 3D object detection model, the height of the scene is a hyperparameter. So, we can not do transfer learning, which means initializing the learnable parameters in the model with the values of parameters from another dataset and training the model with fewer new data. The reason is, when we change the height in the scene, model complexity increases as the scene height increases. Hence, 1x1 convolutional layer was introduced as explained in the section 4.3.2.2. However, the model will perform the detection sometimes as shown in Figure 6.17. Another important observation is, model should be trained each

Hyperparameter	Value
Learning rate	0.001
Learning rate drop	[80,120]
Learning rate decay	[0.0001, 0.00001]
Batch size	2
RPN classification threshold	0.96
Classification loss	binary cross entropy
Regression loss	Smooth L1 loss
Optimizer	SGD/Adam

Table 6.16: Hyperparameters used for training the network.

time on new data-set for proper detection based on the mounting position of LiDAR on the vehicle. The observation is based on the experimentation of model tested on Waymo dataset, where model trained on Kitti dataset.

6.8 Implementation in Finroc

In this section, an overview of the implementation details in the robotic framework of Robotic Research laboratory (RRLAB) is explained. Finroc is a modular C++/Java framework which is used for robotic controls in RRLAB at TU Kaiserslautern, Germany since 2008. In robotic applications, frameworks have an important role and impact on the software quality and therefore, the design of Finroc was motivated by MCA2 architecture. Finroc is characterized by lock-free, zero-copying implementation which also includes support various communication patterns such as queues ([Reichardt 17]). Being highly modular, the concepts/algorithms are written in modules which can be integrated in groups as well as part (which contain their individual interfaces). Finroc is extended by tools such as FINSTRUCT and FINGUI, in which FINSTRUCT handles visualization and connections between ports and FINGUI is a graphical user interface (editor) which assists the projects and can be easily integrated ([Reichardt 12]). An overview of the entire framework can be seen from the figure 6.18.

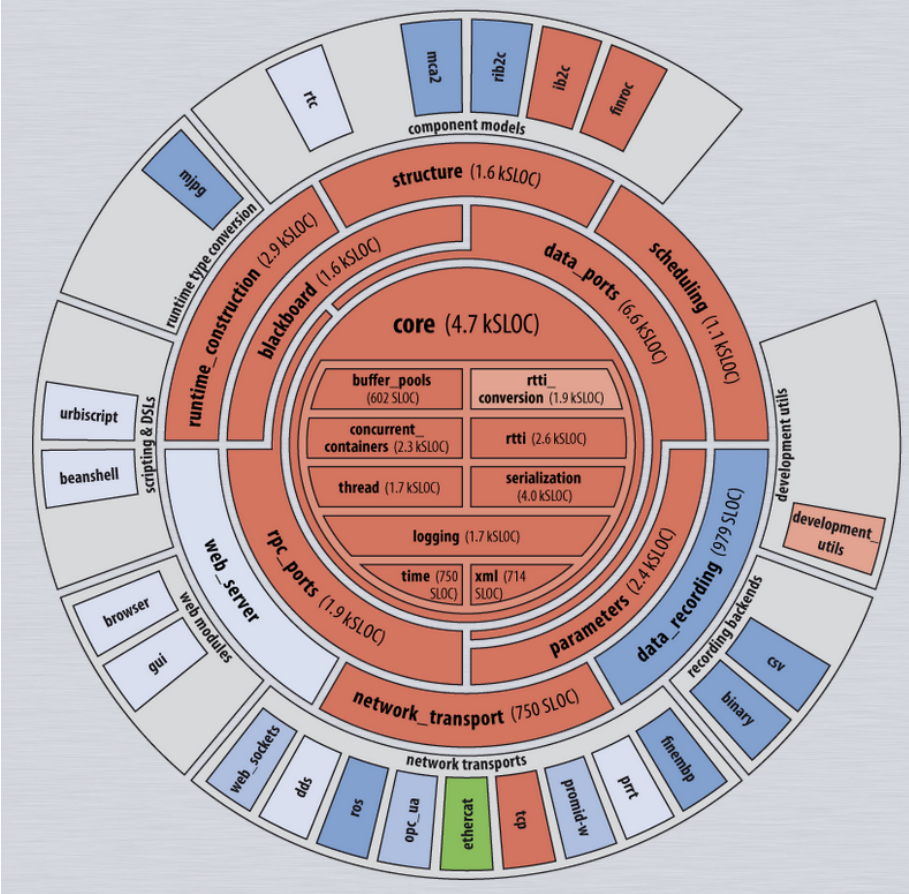


Figure 6.18: FINROC architecture [Reichardt 12]

7. Conclusion

In this section, we will summarize the 3D object detection model and results including the advantages and disadvantages of the chosen model. Furthermore, we will discuss future work, which needs to be considered.

7.1 Discussion and summary

Lots of research has been going on in bringing fully autonomous driving vehicles onto the road. But the problem arises when we need a robust system that can detect far-away objects(maybe objects at a distance of more than 30 meters) like pedestrians and cyclists, and take a necessary control action. Thus, the proposed voxelnet model for LiDAR is only based on 3D object detection without using any camera data. Initially, the voxelnet model is trained and evaluated only for single-class for a distance of 48 m on the Kitti dataset due to the limitation of availability of public datasets and unavailability of open-source LiDAR data annotation tools.

Hence, in our work, we considered voxelnet as a base network and extended the voxelnet model for multi-label 3D object detection, mainly for pedestrians and cyclists and for different distances. We have found that the vanilla voxelnet model has a problem with data augmentation, which reduces the overall accuracy. Thus, a new data augmentation strategy is used along with balanced sampling as explained in 5.1, which in turn increases the mAP of pedestrian detection on the Kitti validation set. When considering both pedestrians and cyclists at the same time, the accuracy of a model is not good as compared to the variants of voxelnet existing today, which will be discussed in 7.2. However, using the proper class imbalance approach, we have attained satisfactory results on both validation and testing datasets of Kitti.

Coming to the LiDAR data annotation tool, We even explored the different data annotation tools available for LiDAR data publicly and found that at present, AWS ground-truth labeler is one of the best annotation tools for LiDAR. Considering the high cost of the AWS annotation tool, we have decided to use another dataset, called Waymo.

Currently, the Waymo dataset is one of the largest datasets publicly available for LiDAR data with different driving scenarios. we have shown the results of the extended model for

single and multi-labelon Waymo in section 6.3. While exploring the Waymo dataset, we have found that the Waymo dataset is not just for object detection, but also for object tracking and the difficulty level of LiDAR data annotation is too high, where even humans cannot classify the object. A simple example is shown in Figure 7.3. As a result, the model gives more false positives, as shown in Figure 6.16. We have tried to remove the frames, which have ground-truth for less than 25% occluded and truncated objects like in Kitti, but due to lack of time, we could not able to do it and check the extended model performance.

Figure 7.1: A simple visualization of difficulty level of occlusion in Waymo dataset.

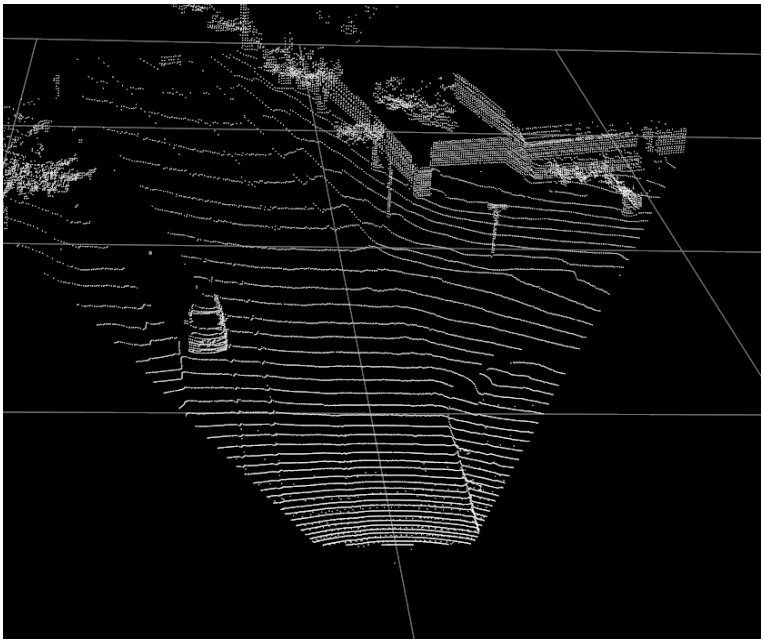


Figure 7.2: A simple visualization of difficulty level of occlusion in Waymo dataset without ground truth.

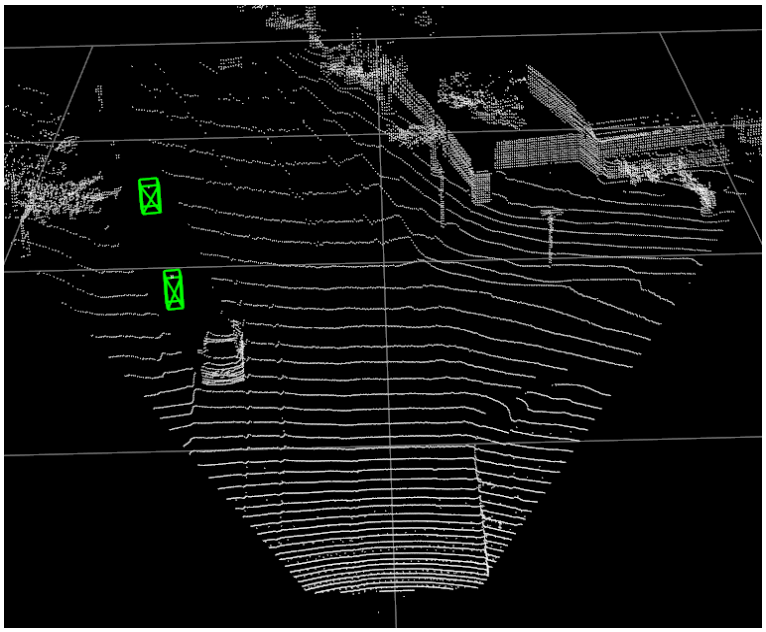


Figure 7.3: A simple visualization of difficulty level of occlusion in Waymo dataset with ground truth.

7.2 Future Work

In this section, the available area of improvement in 3D object detection with LiDAR is presented. Firstly, chosen voxelnet model uses the normal 3D convolutional layer for the 3D feature extractor, which in turn increases the inference time and the need for more training to achieve satisfactory results. To overcome the above-mentioned problem, Benjamin [Graham 15] proposed 3D sparse convolutional layer. In 2020, [Yan 18] proposed a model called SECOND (Sparsely Embedded Convolutional Detection), an extension of the vanilla voxelnet with 3D sparse convolutional as a replacement for 3D normal convolutional for multi-label 3D object detection. The SECOND outperforms all the LiDAR only and LiDAR+RGB based detection networks both on Kitti and Waymo. Along with becoming the SOTA model, also SECOND decreases the inference time to 0.05 seconds per frame, which made the SECOND use in real-time. The basic idea of SECOND is, processing only occupied voxels, instead of unoccupied voxels in 3D sparse convolutional layer and build the voxelgrid of features using scatter operation in Tensorflow after the 3D feature extraction as explained in 4.1.1.4 In 2020, to examine the performance of SECOND on a more sparse LiDAR point cloud and 10 different classes at the same time, [Zhu 19] worked on SECOND with NuScenes [Caesar 20] dataset with different detection heads. The work of [Zhou 18] shows SECOND even outperforms the PP.

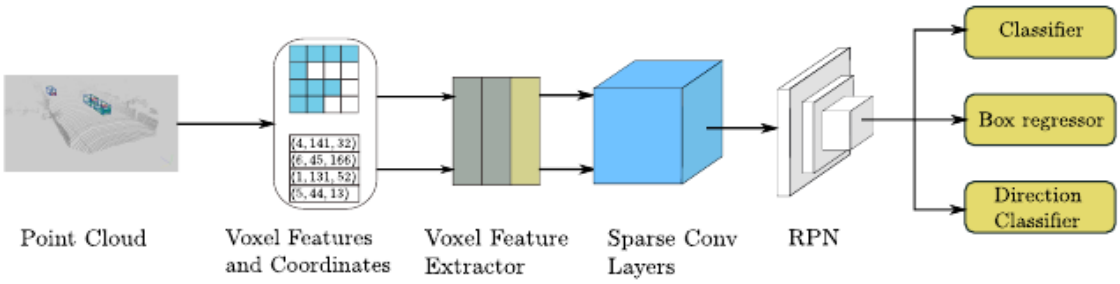


Figure 7.4: An extension of vanilla voxelnet with 3D sparse convolutional network [Yan 18].

Secondly, at the time of my master thesis, the DL framework, TensorFlow did not have the 3D sparse convolutional operation. In March 2021, Tensorflow proposed 3D sparse convolutional custom operation(an attempt was made to use 3D sparse convolutional neural network by building and compiling the custom operation, unfortunately CPU version was built, not the GPU version) as a test version along with a 3D object detection model which is almost similar to SECOND. Therefore, it would be better to use the proposed 3D object detection model in TensorFlow [Rui Huang 21] in near future. An important thing to note, one should make sure that LiDAR preprocessing is implemented on GPU using the Tensorflow framework before using the Tensorflow 3D object detection model.

Thirdly, there is also a possibility to work on the models with different 3D feature extractors like pointnet++ and point pillars.

Finally, to use the Waymo dataset, one has to set the level of difficulty based on the use cases and remove ground-truth of objects, whose difficulty level is below a certain percentage.

Bibliography

[autonomousvision 21] autonomousvision, "LiDAR data annotation tool", 2021 (accessed July 02, 2021). https://github.com/autonomousvision/kitti360Scripts.

[Brownlee 21] J. Brownlee, "A Gentle Introduction to Batch Normalization for Deep Neural Networks", 2021 (accessed July 02, 2021). https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.

[Caesar 20] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[Chen 17] X. Chen, H. Ma, J. Wan, B. Li, T. Xia, "Multi-view 3d object detection network for autonomous driving", in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 1907–1915.

[Du 19] J. Du, "The Frontier of SGD and Its Variants in Machine Learning", *Journal of Physics: Conference Series*, vol. 1229, p. 012046, 05 2019.

[Enthought 21] Enthought, "Data visualizer", 2021. https://github.com/enthought/mayavi.

[Geiger 12] A. Geiger, P. Lenz, R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite", in *2012 IEEE conference on computer vision and pattern recognition*, IEEE. 2012, pp. 3354–3361.

[Girshick 15] R. Girshick, "Fast r-cnn", in *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[Google 15] Google, "Tensorflow framework for deep learning", 2015. https://github.com/tensorflow/tensorflow.

[Graham 15] B. Graham, "Sparse 3D convolutional neural networks", *arXiv preprint arXiv:1505.02890*, 2015.

[He 16] K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[Jaderberg 15] M. Jaderberg, K. Simonyan, A. Zisserman et al, "Spatial transformer networks", *Advances in neural information processing systems*, vol. 28, pp. 2017–2025, 2015.

[Jan 20] Q. H. Jan, "Data set from Robotics research lab", 2020.

[Karpathy 21] A. Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition", 2021 (accessed July 02, 2021). https://cs231n.github.io/neural-networks-1/.

[Ku 18] J. Ku, M. Mozifian, J. Lee, A. Harakeh, S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation", in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE. 2018, pp. 1–8.

[Kumar 20] G. A. Kumar, J. H. Lee, J. Hwang, J. Park, S. H. Youn, S. Kwon, "LiDAR and camera fusion approach for object distance estimation in self-driving vehicles", *Symmetry*, vol. 12, no. 2, p. 324, 2020.

[Lang 19] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12697–12705.

[Liu 16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, A. C. Berg, "Ssd: Single shot multibox detector", in *European conference on computer vision*, Springer. 2016, pp. 21–37.

[Oksuz 20] K. Oksuz, B. C. Cam, S. Kalkan, E. Akbas, "Imbalance problems in object detection: A review", *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[Olof Berg Marklund 21] O. H. Olof Berg Marklund, "3D Object Detection for Autonomous Driving using Deep Learning", 2021 (accessed July 02, 2021). Master's thesis in Computer Science - Algorithms, Languages and Logic.

[open source 21] open source, "Point cloud library", 2021. https://github.com/PointCloudLibrary.

[Openvinotoolkit 21] Openvinotoolkit, "LiDAR/RADAR 3D data labeling", 2021 (accessed July 02, 2021). https://github.com/openvinotoolkit/cvat/issues/1475.

[Phan 20] T. H. Phan, K. Yamamoto, "Resolving class imbalance in object detection with weighted cross entropy losses", *arXiv preprint arXiv:2006.01413*, 2020.

[Pokharna 21] H. Pokharna, "The best explanation of Convolutional Neural Networks on the Internet!", 2021 (accessed July 02, 2021). https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8.

[Qi 17a] C. R. Qi, H. Su, K. Mo, L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[Qi 17b] C. R. Qi, L. Yi, H. Su, L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space", *arXiv preprint arXiv:1706.02413*, 2017.

[Qi 18] C. R. Qi, W. Liu, C. Wu, H. Su, L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data", in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 918–927.

[Reichardt 12] M. Reichardt, T. Föhst, K. Berns, "Introducing finroc: A convenient real-time framework for robotics based on a systematic design approach", *Robotics Research Lab, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, Technical Report*, 2012.

[Ren 15] S. Ren, K. He, R. Girshick, J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.

[Ruder 21] S. Ruder, "An overview of gradient descent optimization algorithms", 2021 (accessed July 02, 2021). https://ruder.io/optimizing-gradient-descent/.

[Rui Huang 21] A. F. Rui Huang, "Tensorflow 3D", 2021 (accessed July 02, 2021). https://github.com/google-research/google-research/tree/master/tf3d.

[Shi 19] S. Shi, X. Wang, H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 770–779.

[Simony 18] M. Simony, S. Milzy, K. Amendey, H.-M. Gross, "Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds", in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018, pp. 0–0.

[Simonyan 14] K. Simonyan, A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv preprint arXiv:1409.1556*, 2014.

[Sun 20] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine et al, "Scalability in perception for autonomous driving: Waymo open dataset", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2446–2454.

[USDT 21] USDT, "Automated Vehicles for Safety", 2021 (accessed July 02, 2021). https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety.

[Versloot 21] C. Versloot, "Max pooling, average pooling", 2021 (accessed July 02, 2021). https://www.machinecurve.com/index.php/2020/01/30/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling/.

[Wang 21] B. Wang, "LiDAR data annotation tool", 2021 (accessed July 02, 2021). https://cs231n.github.io/neural-networks-1/.

[Wikipedia 21 a] Wikipedia, "Sigmoid function", 2021 (accessed July 02, 2021). https://www.google.com/search?channel=fsclient=ubuntuq=Sigmoid+activation+function.

[Wikipedia 21 b] Wikipedia, "Stochastic gradient descent", 2021 (accessed July 02, 2021). https://en.wikipedia.org/wiki/Stochastic$_g radient_d escent$.

[Xu 19] J. Xu, Y. Ma, S. He, J. Zhu, "3D-GIoU: 3D Generalized Intersection over Union for Object Detection in Point Cloud", *Sensors (Basel, Switzerland)*, vol. 19, no. 19, September 2019.

[Yan 18] Y. Yan, Y. Mao, B. Li, "Second: Sparsely embedded convolutional detection", *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[Zhou 18] Y. Zhou, O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4490–4499.

[Zhu 19] B. Zhu, Z. Jiang, X. Zhou, Z. Li, G. Yu, "Class-balanced grouping and sampling for point cloud 3d object detection", *arXiv preprint arXiv:1908.09492*, 2019.