UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Enes Ozipek

Real-time 3D Object Detection on Point Clouds

Master's Thesis (30 ECTS)

Supervisor: Tambet Matiisen, MSc

Tartu 2020

Real-time 3D Object Detection on Point Clouds

Abstract:
The demand for precise and fast object detection frameworks has increased since the autonomous vehicle industry started to attract more attention. While the progress made so far in 2D object detection task with state-of-the-art approaches such as convolutional neural networks seems promising, we still struggle to obtain the same level of performance in 3D modalities such as lidar point clouds. The main reasons are that point cloud is sparse and in 3D while state-of-the-art 2D object detection models work on camera images. Some of the early works have tried to ease the aforementioned challenges using either 3D convolutional neural networks or bird's eye view approaches, nevertheless, they were not able to achieve the desired level of performance in 3D perception.

PointPillars is one of the recent models running fast with a good accuracy on point clouds. Its main advantage arises from the way it encodes the points in pillars into spatial features using PointNet. It basically divides the whole point cloud into grids of vertical pillars and applies state-of-the-art 2D detection network on this top-down view in which spatial features are encoded. Even though this operation enables the network to keep the positional information of the points within each pillar, yet, it does not take into account the point densities in different parts of the point cloud.

This thesis aims to improve PointPillars network by utilizing the positional encoding and extending the detection area. Positional encoding helps the network utilize positional features by introducing two additional input channels before each convolutional and deconvolutional layer. Additionally, different positional encoding schemes are compared to have more insight about the effectiveness of the positional channels introduced. Moreover, this thesis also presents a simple scheme to train 360-degrees model with ground truths provided for only camera Field-of-View (FOV).

Positional encoding scheme provides better accuracy at a similar speed as the original network. On the other hand, even though 360-degrees model is supposedly the type of a model that should be used with lidar, in experiments, it is observed that it outputs many False-Positives (FPs).

Keywords:
Object detection, 3D human detection, Positional encoding, Data augmentation

CERCS:
P170 Computer science, numerical analysis, systems, control

Reaalajas 3D objektituvastus punktipilvedes

Lühikokkuvõte:
Isejuhtivad autod vajavad täpseid ja kiireid objektituvastuse meetodeid. Kaamerapõhises 2D objektituvastuses on tehtud suuri edusamme tänu konvolutsiooniliste närvivõrkude kasutuselevõtule, samas kui samaväärsete tulemuste saavutamine 3D modaalsustes, nagu näiteks punktipilved, on seni jäänud kättesaamatuks. Põhiliseks põhjuseks on, et punktipilved on kolmemõõtmelised ja hõredad, samas kui kaamera pildid on kahemõõtmelised ja tihedad. Varasemalt on katsetatud 3D konvolutsioonide kasutamist punktipilvedel või punktipilve konverteerimist 2D ülevalt-alla vaateks, millele rakendatakse tavalist 2D konvolutsioonilist võrku. Siiski ei ole need saavutanud soovitud tulemusi.

PointPillars on üks hiljuti välja pakutud närvivõrgu arhitektuuridest, mis saavutas punktipilvede peal hea kiiruse ning täpsuse. Selle põhiline eelis tuleneb viisist, kuidas see esitab punktipilve. Kogu punktipilv jagatakse ruudustikuks, mis koosneb vertikaalsetest "sammastest". Iga sammas kodeeritakse PointNeti abiga ruumiliseks vektoresituseks. Sellele ruudustikule rakendatakse kaasaegset 2D objektituvastuse võrku. Kuigi selline lähenemine esitab punktide omavahelisi asukohti ühe samba sees, siis ei võimalda see käsitleda sambaid erinevalt sõltuvalt nende asukohast. See võib olla vajalik, sest punktipilve tihedus väheneb lidarist kaugemal ja seetõttu on vajadus töödelda kaugemaid sambaid erinevalt.

Selles töös püütakse PointPillars võrku parendada tuues sisse asukoha kodeeringu konvolutsioonilistes kihtides ja laiendades tuvastuspiirkonda. Asukoha kodeering võimaldab konvolutsioonilistel filtritel olla teadlik, millises kohas neid rakendatakse. Selleks lisatakse konvolutsioonilise ja dekonvolutsioonilise kihi sisendisse kaks täiendavat asukoha kanalit. Töös võrreldakse ka erinevaid asukoha kodeerimise skeeme. Täiendavalt pakutakse välja lihtne skeem võrgu treenimiseks 360-kraadi ulatuses tuvastuseks kui objektid on märgendatud on ainult kaamera nähtavusulatuses.

Asukoha kodeeringu kasutamine parandas PointPillars võrgu tulemusi, jättes kiiruse praktiliselt samaks. Kuigi 360-kraadine objektituvastus on loomulik lidari jaoks, siis selleks treenitud võrk andis kahjuks palju vale-positiivseid tulemusi.

Võtmesõnad:
objektituvastus, 3D inimeste tuvastus, asukoha kodeering, andmete rikastamine

CERCS:
P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

List of abbreviations

ROS	Robot Operating System
AV	Autonomous Vehicles
UGV	Unmanned Ground Vehicle
LIDAR	Light Detection and Ranging
CNN	Convolutional Neural Network
FOV	Field of View
RPN	Region Proposal Network
SSD	Single-Shot Detector
NLP	Natural Language Processing
BEV	Bird's Eye View
E-RPN	Euler-Region Proposal Network
YOLO	You only look once
PFN	Pillar Feature Net
IoU	Intersection Over Union
FP	False Positive
SIFT	Scale-invariant feature transform
HOG	Histogram of oriented gradients
RNN	Recurrent Neural Network
FPS	Frames per second
NN	Neural Network
FCN	Fully Connected Layer

1 Introduction

Object detection using lidar point clouds is one of the important problems for autonomous driving. Since the robots, more specifically autonomous vehicles (AVs), should perceive the environment around them in 3D, it is preferable to have a network detecting objects in 3D. Furthermore, AVs are expected to operate in real-time so that they can react timely to the environment and moving objects around them. To satisfy those goals, it's essential to deploy a 3D object detection algorithm that is fast and accurate.

Considering the breakthrough achieved by deep learning in computer vision, there is interest in replicating the success also in other modalities, for example, lidar point clouds. However, this poses two problems: compared to the camera images point cloud is sparse and in 3D. Some early works have used 3D convolution [1], others have applied 2D CNNs on bird's eye view [2],[3],[4],[5].Even though 3D Convolutional Neural Networks (CNNs) give a strong performance in the task, they suffer from being slow. On the other hand, the direct application of 2D CNNs is inefficient due to the sparseness of the point cloud. PointPillars [6] is one of the latest network architectures proposed to solve the aforementioned issues. PointPillars creates a grid from the point cloud and considers points in one vertical column as a "pillar". Those columns are passed through PointNet to generate generic 3D features. In the end, it runs a Single-Shot Detector network (SSD) [7] on a pseudo image composed of the learned features.

This thesis focuses on detecting humans from lidar pointclouds. PointPillars has shown remarkable performance in terms of both accuracy and speed for human detection on the KITTI dataset [10]. Even though it is fast and accurate enough for any type of point clouds, there are cases it can make mistakes, for example, misdetecting tree trunks or poles as a human. Furthermore, it is a well-known fact that the point density in a point cloud get sparse farther away from lidar source. Therefore, relying on the features on the pillars introduced by PointPillars might not be sufficient to encode every information available. This work proposes Positional PointPillars, which encodes the position of each grid cell in the network features. The idea is analogous to the CoordConv [9] for 2D convolution or positional encoders used in Natural Language Processing (NLP) transformers [8] . The hypothesis is that positioning of the grid cells in network features can be encoded using different functions such as sine. This would allow the network to adapt its behaviour for expectedly dense or sparse parts of the point cloud, this way preventing misdetections and increasing accuracy.

Positional PointPillars is also an end-to-end network that does not require any hand-tuned feature extractors, instead, it learns to encode all the available information through the network. On the other hand, since the proposed positional encoding scheme does not have many layers and parameters, it is expected to run at a similar speed as original PointPillars.

2 Background

Object detection was an active topic even before Deep Learning phenomena. It was generally performed in several steps starting with edge detection and feature extraction methods such as Scale-invariant feature transform (SIFT), Histogram of oriented gradients (HOG) etc. However, we observed a boost in the advancement of object detection technique with the breakout popularity of CNNs as they were capable of automatically extracting more complex and better features.

2.1 Convolutional Neural Networks (CNNs)

Convolution Neural Networks (CNN) are special type of Feed-Forward Artificial Neural Networks which has three main layers: **Convolution Layer**, **Pooling Layer** and **Fully Connected Layer (FCN)** . The hidden layers of the CNNs are responsible for extracting the features from the input such as images [16]. Convolutional layers employ several number of filters to pass over the inputs and carry out convolution operation to obtain convolved feature maps. Therefore, in each convolutional layers, each filter outputs a feature map and then those feature maps are concatenated to form a single activation map having as many channels as the number of filters. After then this activation map is passed through the next layers and so on.

Convolutional layer convolves, see Figure 1, the input and pass the feature map it creates to the next layer. This feature maps generally has the shape of (feature map width) x (feature map height) x (feature map channels) [17]. It realizes this operation using a set of learnable filters with each filter of size some small width and height. For instance, if the size of an image input is 32x32x3, then typical filter/kernel size can be 5x5x3 as there are three input channels. Furthermore, given that the padding size is 0, stride is 1 and the number filters used in the layer is 16, then the output feature map size will be 28x28x16. Kernels are a significant part of convolutional layers to extract the features such as an edge of some orientation or a blotch of some color on lower layers, or more abstract/general patterns on higher layers of the network.

One of the significant attributes and strengths of the filters is that they are translation invariant. In other words, regardless of the location where a feature exists in an image, the same feature template can be used across the image. Therefore, being able to use the same feature template across the image enables the network to have less number of parameters, which results in easier and faster training. For instance, if we used regular fully connected neural network instead of CNNs, we would have to train huge number of parameters which quickly lead to overfitting.

CNNs commonly use a pooling layer in-between successive in convolutional layers. Its use is to gradually reduce the spatial size of the representation to reduce the number of parameters and computation cost in the network, this way helping to control overfitting. The most common form of a pooling layer has filters of size 2x2 applied with a stride of

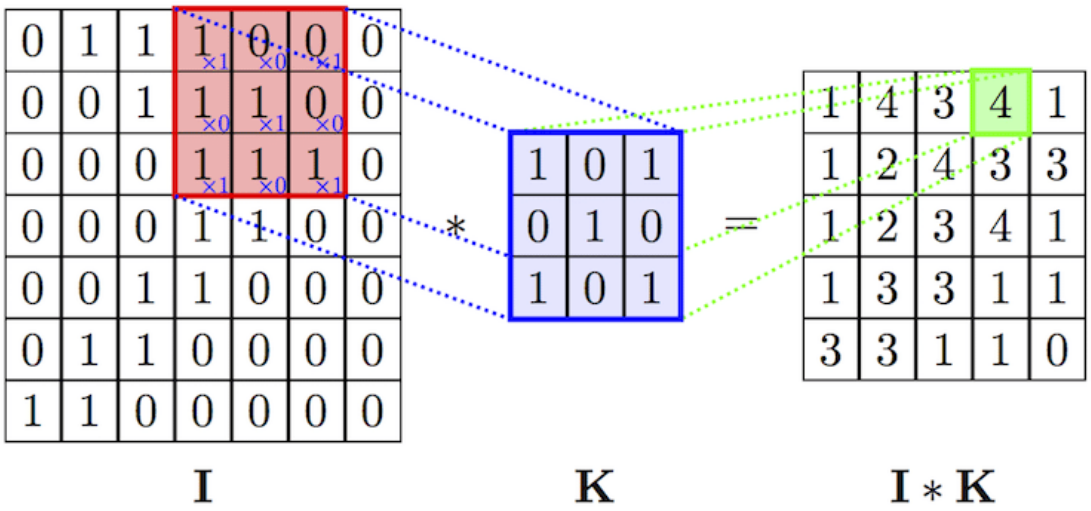


Figure 1. 2D convolution example [29]. The corresponding output value is calculated summing the element-wise multiplication, e.g. $1x1 + 0x0 + 0x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 1x1 = 4$.

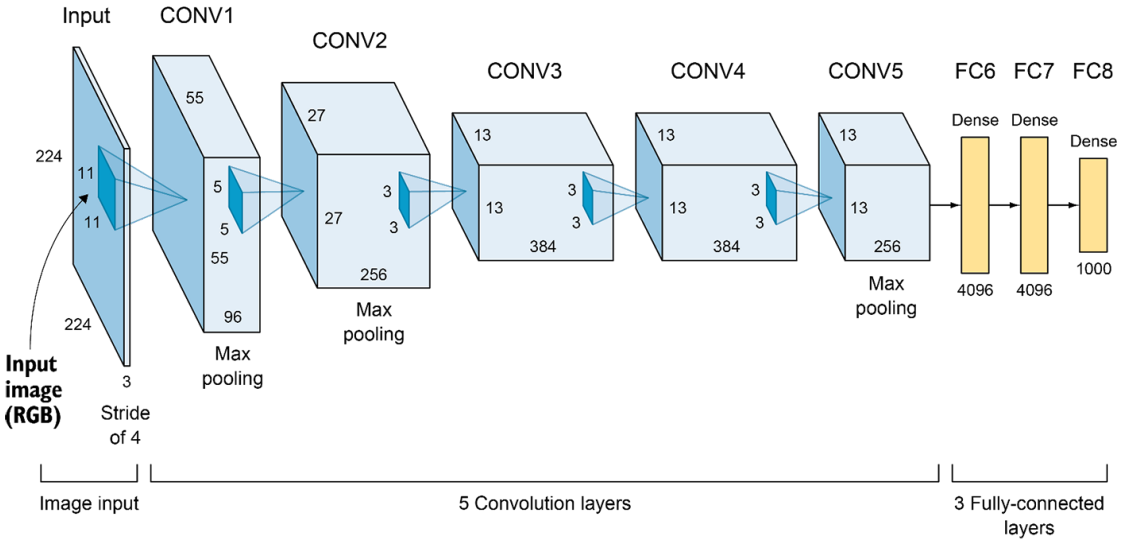


Figure 2. An illustration of the architecture of CNN for AlexNet [18] [28]

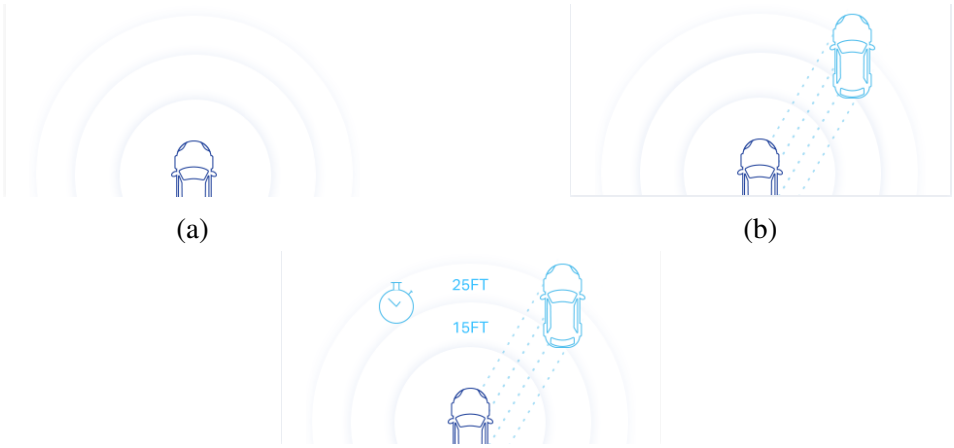


Figure 3. (a) The lidar sensor emits signal (light waves) pulse into the surrounding environment. (b) The lidar sensor catches the pulses bounced off surrounding objects. (c) The lidar sensor measures the time it takes for each pulse to come back to the sensor, hence calculates the distance [19].

2, which downsamples the input by 2 along both width and height.

On the other hand, fully connected layers connect each neuron in one layer to each neuron in another layer. In principle, it works the same way as the traditional multi-layer perceptron neural network (MLP). The flattened output of the previous layer goes through a fully connected layer to classify the images, see Figure 2.

2.2 LiDAR

LiDAR (LIght Detection And Ranging) is an active remote sensing system which itself generates energy, light, to measure surroundings on the ground. It uses eye-safe laser beams to create a 3D representation of the environment.

Lidar sensor emits signal into the surrounding to map the environment in 3D, see Figure 3. All lidar data points will have an corresponding X, Y location and Z (elevation) values. Furthermore, they have an intensity value, which demonstrates the amount of light energy recorded by the sensor. One of the most common way of representing the lidar data is point cloud due to the simplicity of accessing the handy libraries and software tools. You may consider Figure 4 to understand how an example point cloud scene recorded by Velodyne 64-beam lidar sensor looks like.

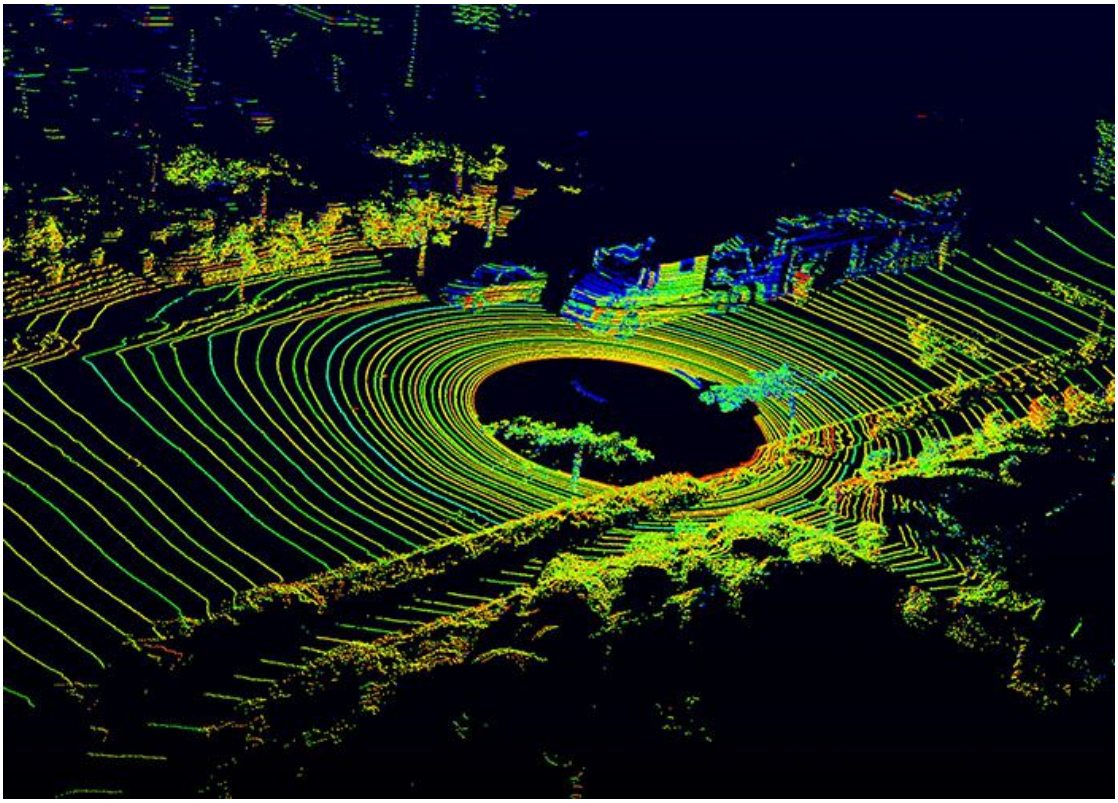


Figure 4. Velodyne 64-beam lidar point cloud [30]

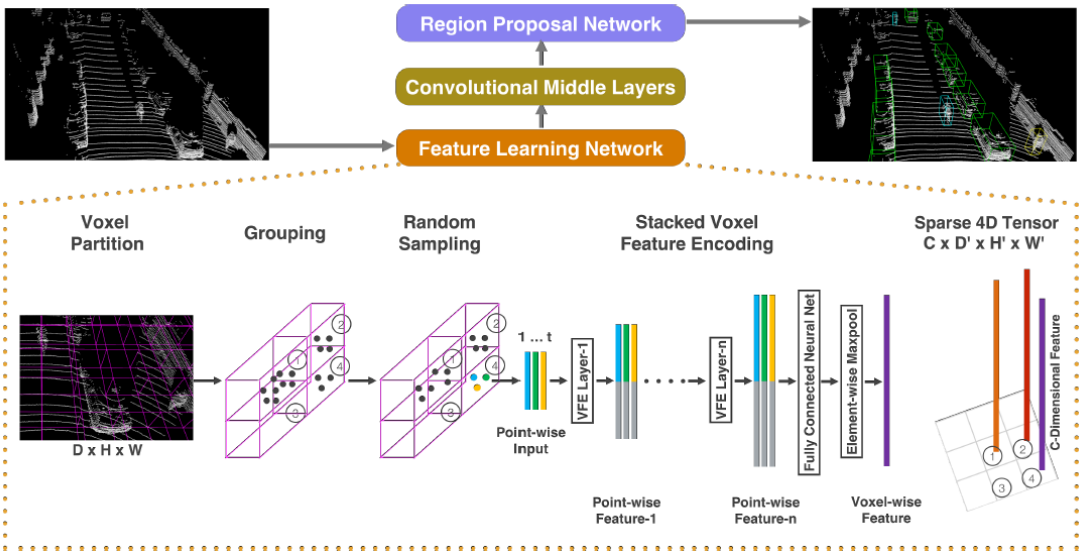


Figure 5. VoxelNet [4] architecture. The feature learning network takes a raw point cloud as input, partitions the space into voxels, and transforms points within each voxel to a vector representation characterizing the shape information. The space is represented as a sparse 4D tensor. The convolutional middle layers processes the 4D tensor to aggregate spatial context. Finally, a RPN generates the 3D detection.

2.3 3D Detection on Pointclouds

Object detection in 3D has been gaining more and more attraction since the autonomous vehicle industry started engaging the lidar. Some of the prior works have used lidar-only methods [4],[5],[6],[12], others have used camera-fused approaches [13],[14]. Lidar-only methods subdivided into two according to encoder type they used: *fixed* and *learned*.

VoxelNet [4], see Figure 5, is one of the pioneer works of end-to-end learning for 3D object detection on point clouds. First of all, the point cloud is divided into equally spaced voxels. The points in each voxel are grouped together and augmented. Each point in each voxel has four features in the beginning: coordinates x, y, z, and *intensity*. Then, each point is augmented by three more features: x_c, y_c, z_c where c subscript denotes distance to the arithmetic mean of all points in the voxel. This compact representation of 3D space is passed through an encoding layer which transforms the points within each voxel into a feature tensor, this way ensuring that VoxelNet does not require manual feature engineering and it learns encoding. Secondly, a 3D convolution layer is applied to the input tensor to consolidate features output by the encoding layer. The Region Proposal Network (RPN) produces a probability score map and a bounding box regression map over the output of the Convolutional Middle Layers. Even though the VoxelNet provides good results with end-to-end learning, it suffers from being too slow for real-time.

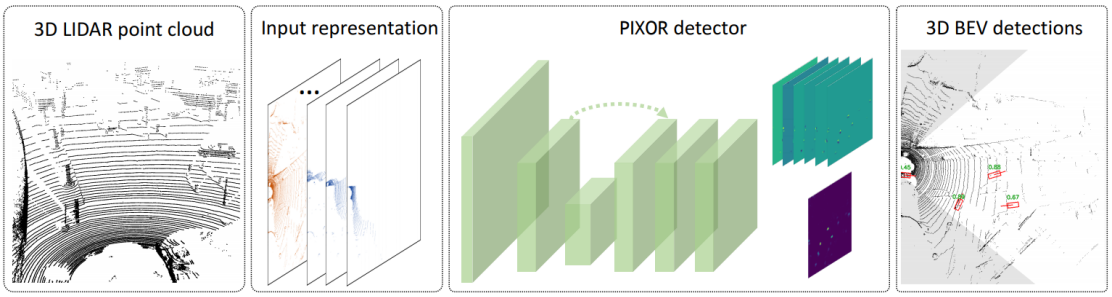


Figure 6. Overview of the proposed, PIXOR [5], 3D object detector from Bird's Eye View (BEV) of LIDAR point cloud. 3D point cloud converted into BEV. PIXOR detector extracts general representation of the input in the form of convolutional feature maps and outputs detections pixel-wise.

PIXOR [5], see Figure 6, alleviates the dependency on 3D convolutional layers by representing the whole point cloud from the Bird's Eye View (BEV). It is a single-stage, LiDAR-only model for 3D localization of vehicles. Its main contribution is to change the representation of the lidar data. The authors of the paper suggest discretizing 3D space into a 3D voxel grid of resolution 0.1 m, each grid cell is then either assigned a value of one or zero, denoting if there is any point in the corresponding grid cell. This representation leads to the 3D occupancy map with 35 channels (3.5 m height of resolution 0.1m) in total. However, they also add a intensity channel along with two additional channels to occupancy feature maps to cover out-of-range point, which results in 38 channels in total. PIXOR detector outputs predictions pixel-wise with each prediction corresponding to a 3D bounding box as it then assigns an anchor per grid cell (pixel) and uses non-max-suppression (NMS) to discard wrong detections. After NMS is applied, detections are decoded into 3D representation.

Even though PIXOR contributes to the 3D detection problem with their novel idea and manages to have a good performance, one of the main drawbacks of this network is that it encodes the height with fixed-sized bins. This manual encoding gives poor results with different BEV map resolutions, especially at larger ones.

ComplexYOLO [12], see Figure 7, is also one of the works which makes use of the BEV approach to encode the point cloud. The R channel refers to the height, G channels to the intensity and B channel to the density of the point data, the resolution of the grid map is 8cm and a constant height of 3m is used for all the objects. Euler-Region-Proposal network (E-RPN) then predicts five objects per grid along with regression parameters, an angle, which is not present in 2D Yolo algorithm, objectness score and class probabilities. The idea is analogous to the existing 2D Yolo algorithm [31]. Even though it works quite fast, it uses fixed encoders and fails to yield better results than the recent methods.

PointPillars [6], see Figure 8, is one of the recent 3D lidar-only object detection algorithms which provides good performance at real-time speed. Therefore, it can be

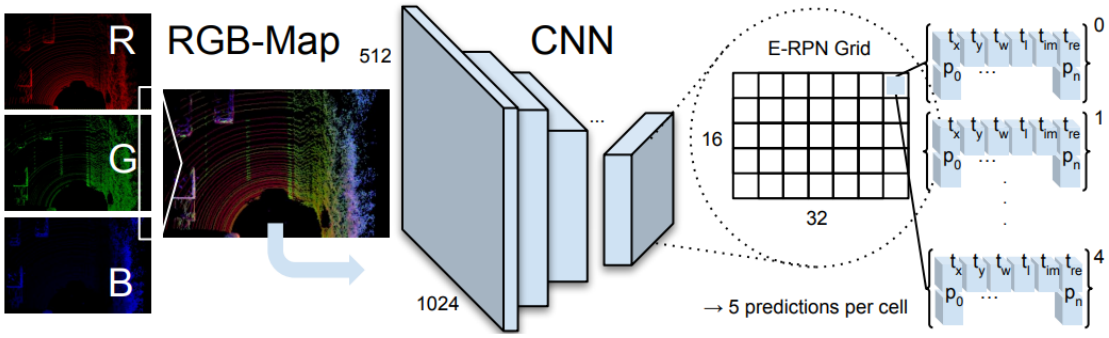


Figure 7. ComplexYOLO [12], feeds the RGB-map into the CNN. The E-RPN grid runs simultaneously on the last feature map and predicts five boxes per grid cell. Each box prediction is composed by the bounding box regression parameters *t* and object scores *p* with am objectness probability p_0 and *n* class scores $p_1 \dots p_n$. The angle is calculated using t_{im} and t_{re}, i.e $\phi = \arctan(t_{im}, t_{re})$.

considered a viable option when it comes to real-time applications such as Unmanned Ground Vehicles (UGVs).

Pillar Feature Net (PFN) is the novel part introduced by the authors. For some specific range, and resolution, the whole point cloud is represented as a top-down grid map and pillar is created from points in each grid cell. Each point in each pillar has four features in the beginning: coordinates *x*, *y*, **z**, and *intensity*. Then, each point is augmented by five more features: x_c, y_c, z_c, p_x, p_y where *c* subscript denotes distance to the arithmetic mean of all points in the pillar and the *p* subscript denotes the offset from the pillar *x*, *y* center. Each point augmented becomes 9-dimensional. After constituting a compact representation of the raw points, this input representation goes through an embedding layer (PointNet [15]) to convert the raw point information into point features. This representation goes through a max operation over the channels, which outputs learned features as given in Figure 8.

In the final step of the PFN Layer, learned features are mapped to a pseudo image of size with given height *H* and width *W*. This mapping operation requires the indices of pillars created during the point augmentation part, that is, when the pillars are stacked together. The final pseudo image output by PFN Layer is passed to the Backbone part of the network architecture, see Figure 8.

The Backbone part consists of three downsampling and upsampling blocks. Each downsampling block has more than one 3x3 2D Conv-layers. After each downsampling operation, features are upsampled using ConvTranspose2d. Then upsampled features are concatenated along with the channels. Concatenated feature map is passed to the Detection Head of the network to produce 3D bounding box detections for objects.

SSD part of the network matches the prior boxes to the ground truth using 2D

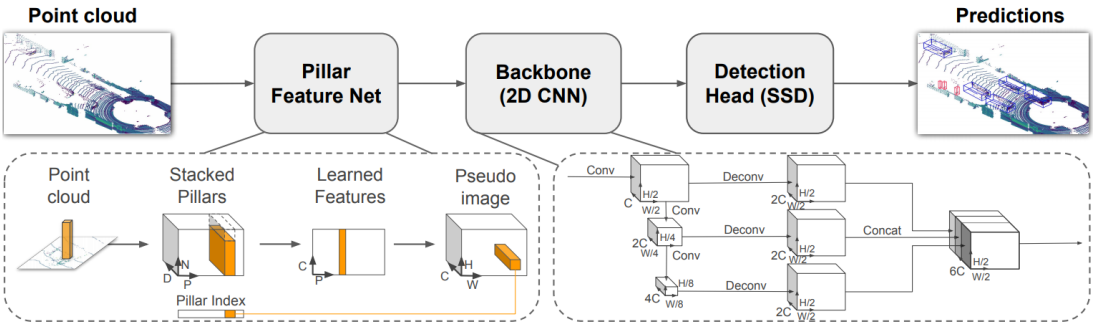


Figure 8. The main components of the PointPillars' network [6] are a Pillar Feature Network, Backbone, and SSD Detection Head. The raw point cloud is converted to a stacked pillar tensor and pillar index tensor. The encoder uses the stacked pillars to learn a set of features that can be scattered back to a 2D pseudo-image for a convolutional neural network. The features from the backbone are used by the detection head to predict 3D bounding boxes for objects

Intersection over Union (IoU) [21]. Furthermore, there are two anchor boxes defined by a width, length, height, and z center for each class. They are applied using two orientation angles: 0 and 90 degrees. Anchors are used for matching ground truth using the 2D IoU according to the the following rules:

- A matching is considered positive if the IoU between the anchor and the ground is either above the positive threshold or is the highest with a ground truth box.
- A matching is considered negative if the IoU between the anchor and the ground is below the negative threshold.
- All other anchors are discarded during training.

During inference, axis aligned NMS is applied with the threshold of 0.5 IoU. The reason for that it gives comparable performance as the rotational NMS, yet, it is much faster.

The first main problem with the existing structure of the PointPillars is that it does not explicitly provide the network with the positional information of the learned features through RPN. In other words, standard convolutional layers do not let the filters know where they are on the pseudo image or the feature map. This thesis seeks a way of alleviating the aforementioned problem to improve accuracy.

The second problematic case is that the dataset on which it is trained has ground truths only in camera FOV. This issue makes the network to be inclined to work solely on FOV. The reason is that the data distribution differs from a sector to another, see Figure 9.

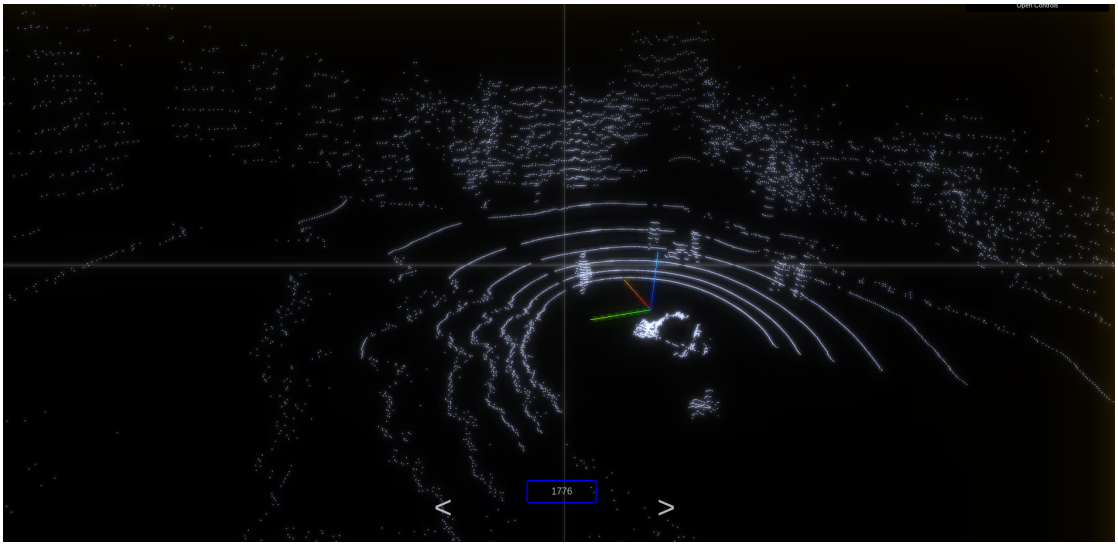


Figure 9. Human points in front of and behind the UGV. Convex arches point outwards in different directions.

A simple solution to make the network work on all of the sectors is to rotate the point cloud and run the model three more times.

2.4 Positional Encoding

Convolutions are omnipresent in today's modern deep learning architectures. One of the significant attributes and strengths is that they are translation invariant. In other words, regardless of the location where a feature exists in an image, the same feature template can be used across the image. As discussed in subsection 2.1, being able to use the same feature template across the image enables the network to have less number of parameters, which results in easier and faster training. Nevertheless, translational invariance might not always help the convolutions achieve every tasks. Uber AI Labs explains the intriguing failure of CNNs [9] using three example problems:

- **Supervised Rendering:** Given some Cartesian coordinate (i,j), highlight a 9x9 patch on a 64x64 canvas centered around the provided coordinate.
- **Supervised Coordinate Classification:** Given some Cartesian coordinate (i,j), provide its corresponding one-hot vector representation for a matrix of size 64x64.
- **Supervised Regression:** Given a 64x64 one-hot matrix containing a single white pixel, provide its corresponding Cartesian coordinate representation in the 64x64 space.

The solution they propose is to use CoordConv layer, see Figure 11, instead of plain convolutional layer. CoordConv basically allows filters to know where they are in Cartesian space by adding additional, hard-coded input channels containing coordinates of the data seen by the convolutional filter.

There are several ways of encoding positional information using additional channels of CoordConv. One of the simplest example of encoding style is *linear* as proposed by Uber's AI Labs [9]. For a 2D image of size (WxH), with H rows and W columns, two additional input channels, see Figure 10, are added to the original input with the following properties:

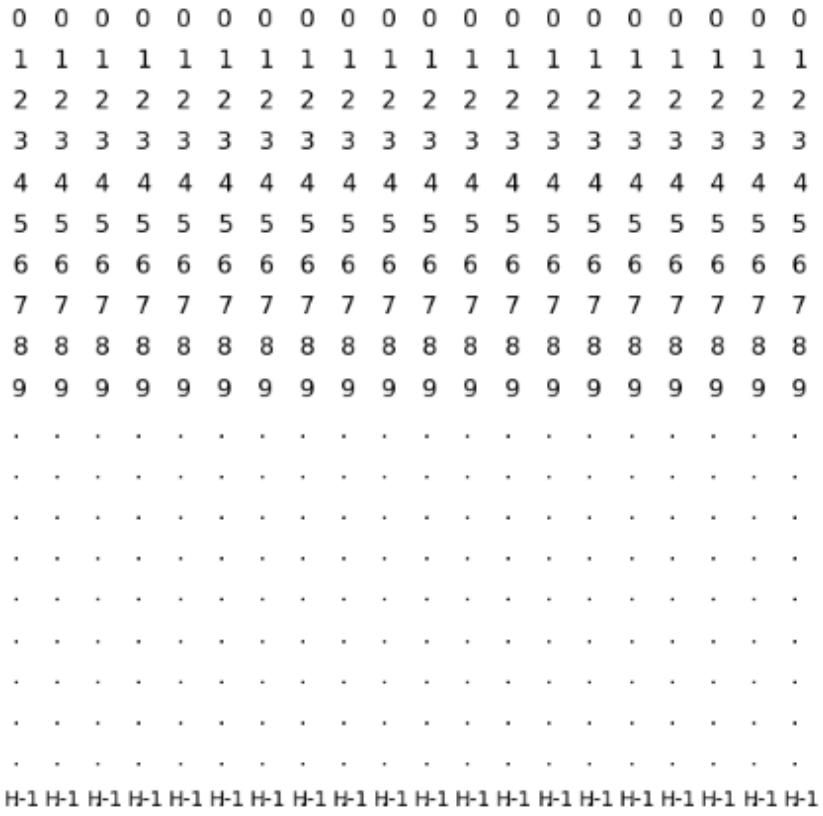
- For the first channel, 0th row contains only 0s, the next row contains only 1s, the next one contains only 2s and the last row contains only (H-1)s etc.
- For the second channel, 0th column contains only 0s, the next column contains only 1s, the next one contains only 2s and the last column contains only (W-1)s etc.
- Both channel values are normalized onto the range [-1,1].

This is how pixel coordinates are encoded at the input layer given that input is a type of image. However, CoordConv is not restricted to work with only images or at the input layer. It can be functional in the intermediate layers, since, for other layers in the network, the number of rows and columns will be corresponding to the height and width of the feature map at that particular layer. Therefore, CoordConv can be applied at any layer where you think letting the filters know their positions would be helpful for the network in general.

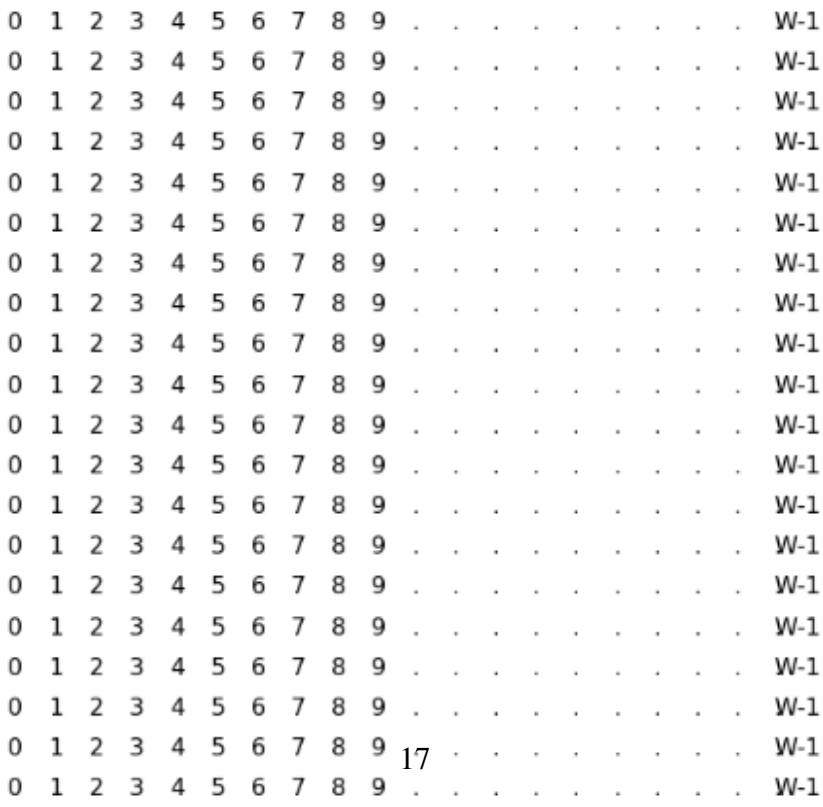
Additional channels increase the number of parameters. However, the increase in the number of parameters can be calculated and fixed beforehand according to the explanation provided by the authors of the paper:

> A standard convolutional layer with square kernel size k and with c input channels and c' output channels will contain $c \times c' \times k^2$ weights, whereas the corresponding CoordConv layer will contain $(c+d) \times c' \times k^2$ weights, where d is the number of coordinate dimensions used (e.g. 2 or 3), see Figure 11.

Moreover, it seems that the increase in number of parameters should not affect the performance too much. In this work, it is measured that base PointPillars model has 4823340 parameters while CoordConv models have 4874796 parameters where the number of additional parameters introduced due to the CoordConv layer is 51456. Feature map sizes are also the same with respect to the width and height dimensions except the number of channels as CoordConv layers provides the input with two additional channels for position encoding.



(a) Row channel



(b) Column channel

Figure 10. CoordConv channels introduced for linear positional encoding

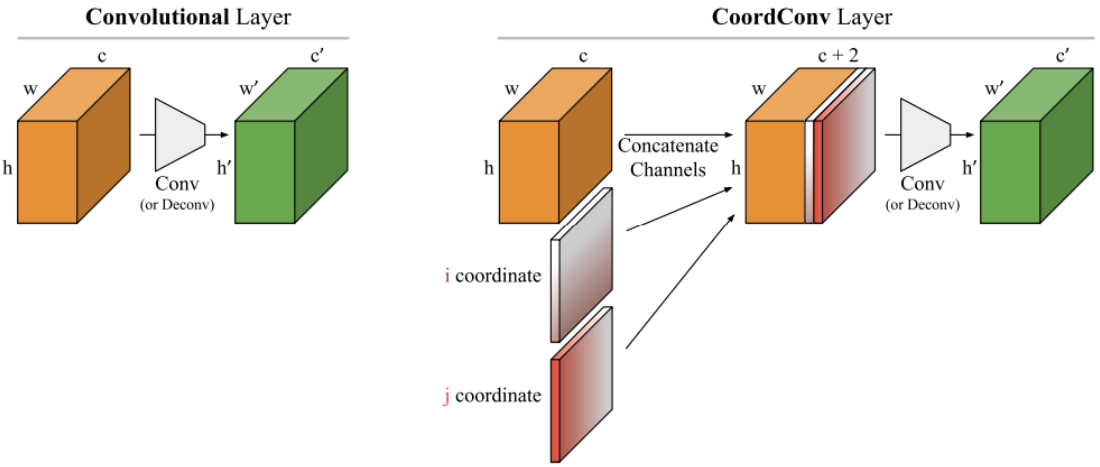


Figure 11. Comparison of 2D convolutional and CoordConv layers [9]. (left) A standard convolutional layer maps from a representation block with shape $h \times w \times c$ to a new representation of shape $h' \times w' \times c'$. (right) A CoordConv layer has the same functional signature, but accomplishes the mapping by first concatenating extra channels to the incoming representation. These channels contain hard-coded coordinates, the most basic version of which is one channel for the i coordinate and one for the j coordinate, as shown above.

On the other hand, it is still possible for the network to not make use of additional channels if the network does not benefit from them. In other words, since the coordinate data are introduced using extra concatenated channels, the convolutional kernels can still figure out to ignore these if they are not helpful for the task the network is being used to resolve. Actually, if the network learns to assign weights of 0s to the parameters used over these channels, then the filters will behave as though we use normal convolutions rather than CoordConv. We can consider the normal convolution input as a subset of CoordConv, one that can be learned if appropriate for the task.

2.4.1 Transformers

One of the specific attributes of the Transformer architecture [8] proposed by Vaswani as an attention-only sequence-to-sequence framework is the positional encoding. The authors propose using $\sin and \cos$ functions to encode the positions of the words in a sentence.

In NLP tasks, the order and the position of the words in a sentence can make considerable differences as they can bring out mistakes in terms of semantics and grammar. Recurrent Neural Networks (RNNs) actually take into account the order of the words in a sentence. However, authors favor the Multi-Head Self-Attention mechanism,

see Figure 13, to speed up the training process and enable the network to figure out the longer dependencies in the sentence [22].

The words go through the encoder and decoder stack of the Transformer's architecture, yet, they are not provided with the positional information. At this point, positional encoding mechanism comes into play. According to the authors, this encoding mechanism should bear the following properties [22]:

- It should provide each time-step with a unique encoding.
- There should be a consistent distance between any two time-steps.
- It should generalize well for different length of inputs and be deterministic.

Therefore, $\sin and \cos$ are a good candidate function satisfying aforementioned criteria. Equation 1 models the encoding scheme for the specific position in a sentence where the encoding forms a vector of size *dmodel*.

$$PE := \begin{cases} PE(pos, 2i) = \sin(pos/10000^{2i/dmodel}) \\ PE(pos, 2i+1) = \cos(pos/10000^{2i/dmodel}) \end{cases} \tag{1}$$

where *pos* is the position and *i* is the dimension. In other words, each dimension of the positional encoding corresponds to a sinusoid.

From the perspective of this thesis, sinusoidal encoding is is to show the feature locations can alternate in a grid bit by bit. Imagine numbers from 0(0000) to 15(1111) in binary format. If you consider the rate of change in the bits, you notice that the least significant bit alternates on each number, the second one changes on every two numbers, and so on. As we deal with floating numbers in neural networks, we use continuous counterparts of the alternating bits, that is, sinusoidal functions.

0 :	0	0	0	0	
1 :	0	0	0	1	
2 :	0	0	1	0	
3 :	0	0	1	1	
4 :	0	1	0	0	
5 :	0	1	0	1	
6 :	0	1	1	0	
7 :	0	1	1	1	
8 :	1	0	0	0	
9 :	1	0	0	1	
10 :	1	0	1	0	
11 :	1	0	1	1	
12 :	1	1	0	0	
13 :	1	1	0	1	
14 :	1	1	1	0	
15 :	1	1	1	1	

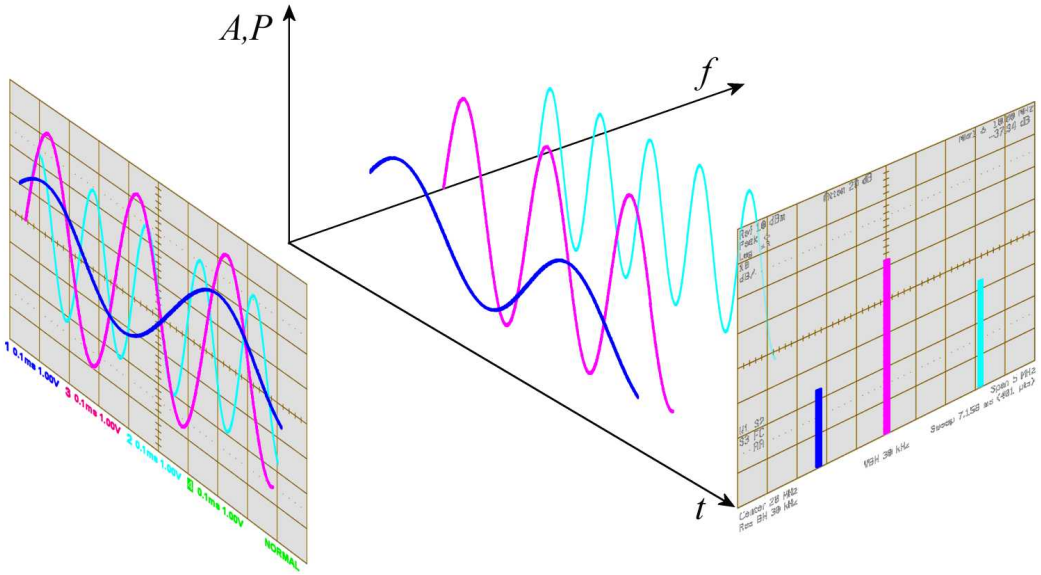


Figure 12. Three-dimensional representation of the viewing directions for the time domain and the frequency domain [36]. Axes of the figure represents the amplitude/power (A,P) of the signal, time (t) and frequency (f) respectively.

For instance, consider Figure 12. It demonstrate the same sinusoidal waves in time and frequency domains. While blue colored wave has the lowest frequency, cyan colored wave has the highest frequency. As it can be observed, sinusoidal waves vary, or cycle, regularly at different rates. These waves are just an example to be made use of by sinusoidal positional encoding.

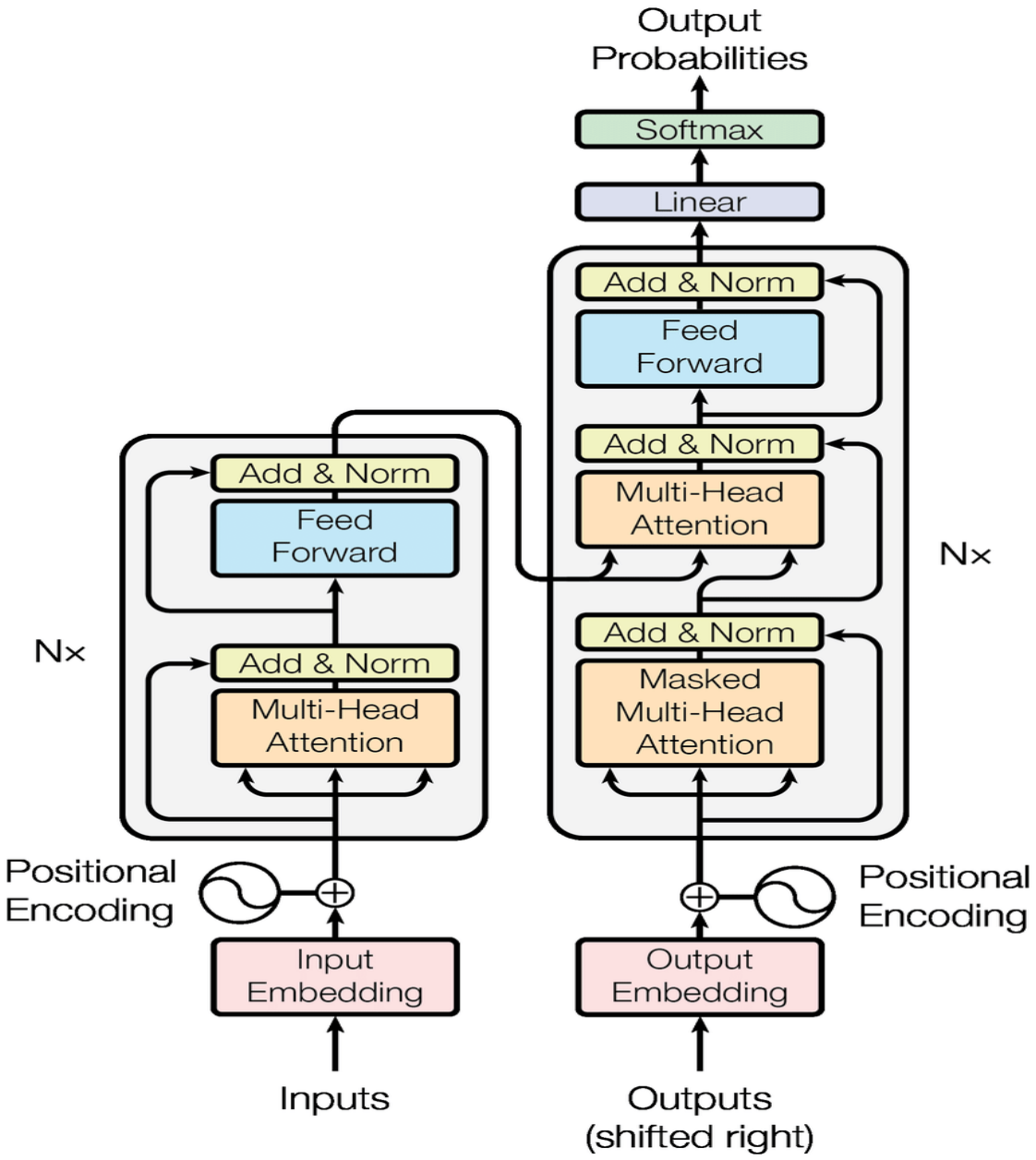


Figure 13. The Transformer - model architecture [8]. The Encoder parts is on the left and the Decoder part is on the right. Both Encoder and Decoder have modules that can be stacked on top of each other multiple times (Nx). The inputs and targets are first embedded into an n-dimensional space as strings can not be used directly. Another significant part of the model is the positional encoding of the different words. Since the architecture does not contain recurrent networks that keep the sequence of words, every word in a sequence is given a relative position owing to the fact that a sequence depends on the order of its elements. These positions are added to the embedded representation (n-dimensional vector) of each word [32].

3 Methods

This chapter discusses the two main experiments performed besides training a baseline model using different datasets, namely KITTI and Milrem. The first experiment is to provide the convolutional layers in RPN with two additional channels so that filters can be aware of their locations. This positional information given to filters is expected to help increase precision metrics. The second experiment is to achieve 360-degree detection using two approaches: replicating the FOV during *training* and *inference*.

3.1 Datasets

Experiments are made on two different datasets: *KITTI* and *Milrem* using 3D ground truths.

3.1.1 KITTI

KITTI object detection benchmark dataset [10] consists of samples that have both lidar point clouds and images. In this work, only lidar point clouds, see Figure 14, are used for training. KITTI originally has 7481 training and 7518 testing samples. For having comparable results with PointPillars, this study splits the official training and validation dataset into 6697 training samples and 784 validation samples, The KITTI benchmark ranks the models for detections of cars, pedestrians, and cyclists. Since the main interest of the thesis is to train a model on the Milrem dataset, KITTI dataset is used to train a model for cars only.

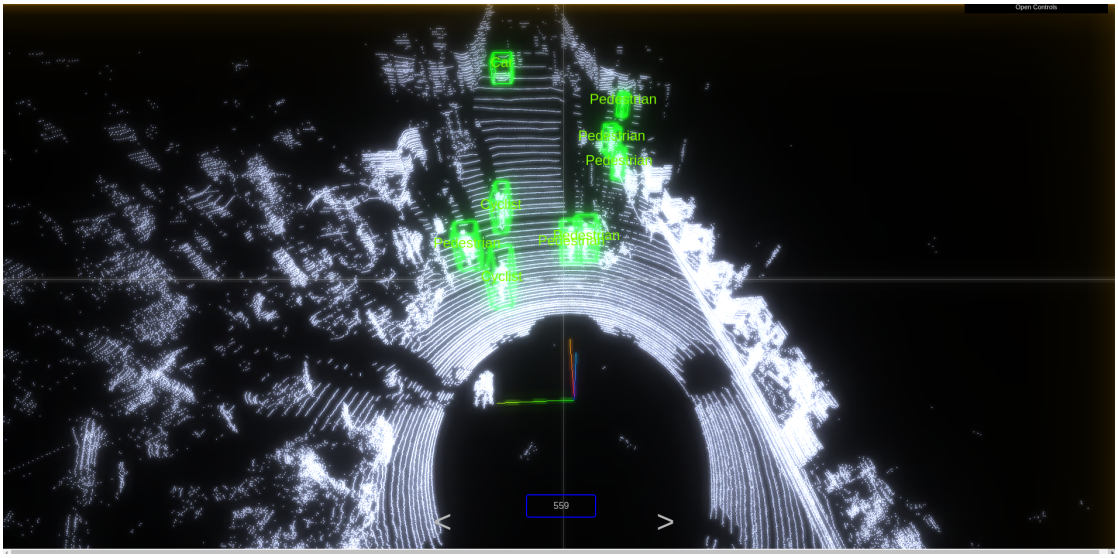


Figure 14. A lidar point cloud sample, from the KITTI dataset, containing cars, pedestrians and cyclists. Ground truths exist only in camera FOV.

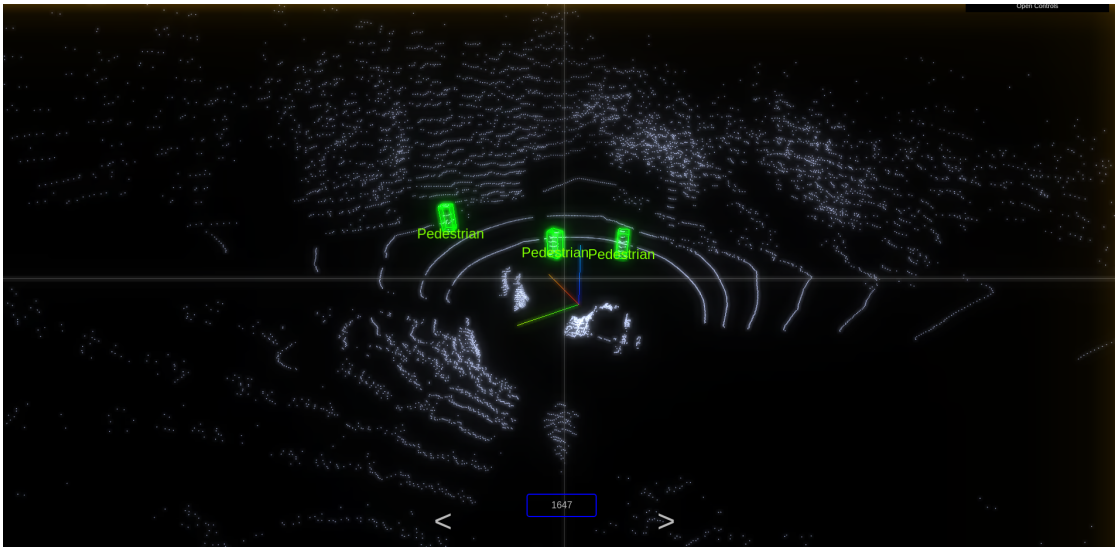


Figure 15. A lidar point cloud sample, from the Milrem dataset, containing pedestrians. Ground truths exist only in FOV.

3.1.2 Milrem

Milrem dataset also consists of samples that have both lidar point clouds, see Figure 15, and images. However, it has 3D ground truths for only pedestrians. Milrem dataset has 3828 samples in total. For this study, it is split into 3478 training samples and 350 validation samples. On the Milrem dataset, a lidar-only model is trained for pedestrian class.

3.1.3 Evaluation Metrics

In this thesis, KITTI evaluation is used to evaluate the models quantitatively. It evaluates 3D object detection performance using the PASCAL criteria also used for 2D object detection. According to the PASCAL criteria [23], the detection task will be judged by the precision/recall curve. The principal quantitative measure used will be the average precision (AP) which is measured using Precision-Recall curve.

Precision measures the ratio of the true object detections to the total number of detections predicted by the model. Good precision value denotes that whatever object the model is predicting, there is a high likelihood that whatever the model detects as a positive detection is a correct prediction. *Recall* measures the ratio of the true object detections to the total number of objects in the dataset. Good recall value denotes that most of the objects in the dataset are detected by the model.

To calculate AP for a specific class, see Figure 16, precision and recall values are calculated using different score thresholds that determines what is considered as a model-

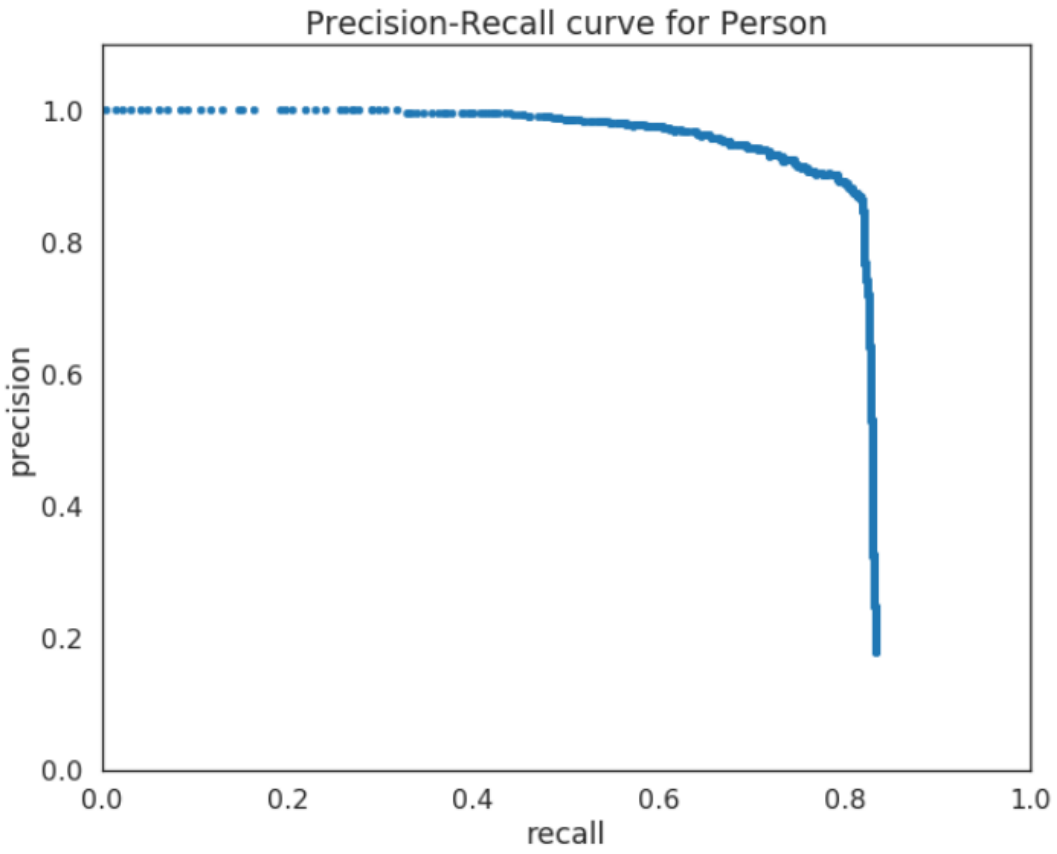


Figure 16. Precision and Recall illustrated [34].

predicted positive detection of the class. Finally, AP is calculated taking the average of precision values all across evenly spaced 11 recall values.

Far objects are filtered out based on their bounding box height in the image plane. Furthermore, the evaluation does not automatically handle detections that are not visible on the image plane - these detections might give rise to false positives. For cars, it requires an 3D bounding box IoU of 70%, while for pedestrians and cyclists it requires a 3D bounding box IoU of 50%. IoU is a measure of overlap between two bounding box; detection and ground truth boxes. In computer vision it is used for correctly detecting an object and calculated as in Figure 17. According to the KITTI evaluation, difficulty levels of boxes are defined as follows:

- **Easy:** Min. bounding box height: 40 Px, Max. occlusion level: Fully visible, Max. truncation: 15 %
- **Moderate:** Min. bounding box height: 25 Px, Max. occlusion level: Partly occluded, Max. truncation: 30 %

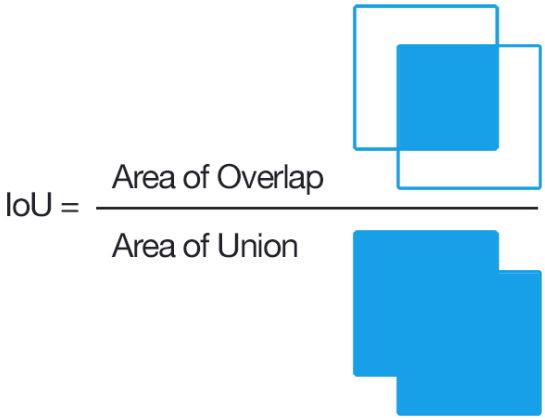


Figure 17. Intersection over Union is the ratio of the area of overlap between the bounding boxes to the area of union [35].

- **Hard:** Min. bounding box height: 25 Px, Max. occlusion level: Difficult to see, Max. truncation: 50 %

However, difficulty levels are determined while dataset is labelled, therefore, they might not be suitable for every datasets available.

There are four main criteria in KITTI evaluation: BBOX, BV, 3D and AOS. While BBOX is used for evaluating the 2D object detection after projecting the 3D bounding boxes to the image, 3D is used for evaluating the 3D bounding boxes. BEV is used to evaluate the predictions using bird's eye view approach. Average orientation similarity (AOS) takes into account the orientation of the predicted boxes. We do not use this as a performance metric in this thesis.

3.2 Positional Encoding

Positional encoding, as explained in subsection 2.4, allows the CNN filters to know where they are in Cartesian space by adding additional, hard-coded input channels containing coordinates of the data seen by the convolutional filter. These experiments aim to show that CoordConv layer can indeed help improve accuracy of the model using different positional encoding schemes: *Linear*, *Sinusoidal*, *Gaussian*. For the sake of having a reliable comparison between the different positional encoding schemes, the same parameter values such as voxel size, number of filters etc. are used to train the models for the same number of epochs.

3.2.1 Linear Encoding

Linear encoding scheme is the way Uber AI Labs implement in their paper. According to the Figure 11 on the right, feature map has height h and width w, and the channel number is c. As explained before, the first channel corresponding to the ith channel in the figure keeps the row coordinates in its rows. Then those coordinates are normalized between zero and one by dividing the tensor with height h. The last operation before the concatenating additional channels with the input tensor is to multiply coordinate values with two and subtract one. This makes sure that all of the coordinate values fall into the range between -1 and 1.

Putting the range between -1 and 1 is often desired for an input when we work with images using neural networks. Sticking to this rule can be a handy tool in 3D object detection task with PointPillars as it encodes the point cloud into a pseudo image. For the second channel, all of the operations are analogues to ones made for the first channel. However, this time, coordinate values should be divided with width w.

3.2.2 Sinusoidal Encoding

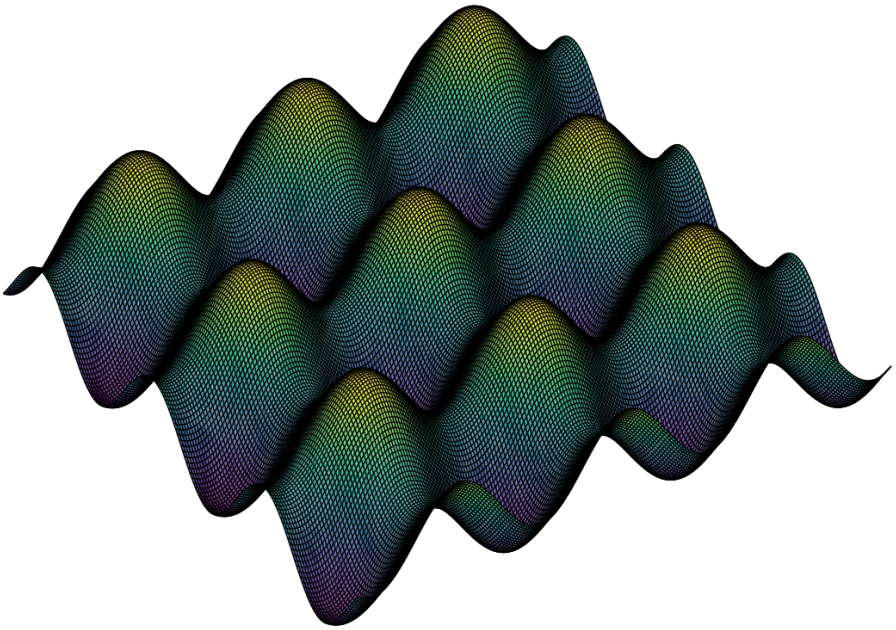
Trigonometric functions are called periodic functions because they repeat over a given period or frequency. This experiment is realized to observe how helpful the bumps generated by a sine function could be for the filters to figure out their locations, hence, the most useful features in a feature map, see Figure 18. In this experiment a range of frequencies between 10 Hz and 20 Hz is tried. The choice is based on how frequently the bumps are observable in the grid map.

3.2.3 Gaussian Encoding

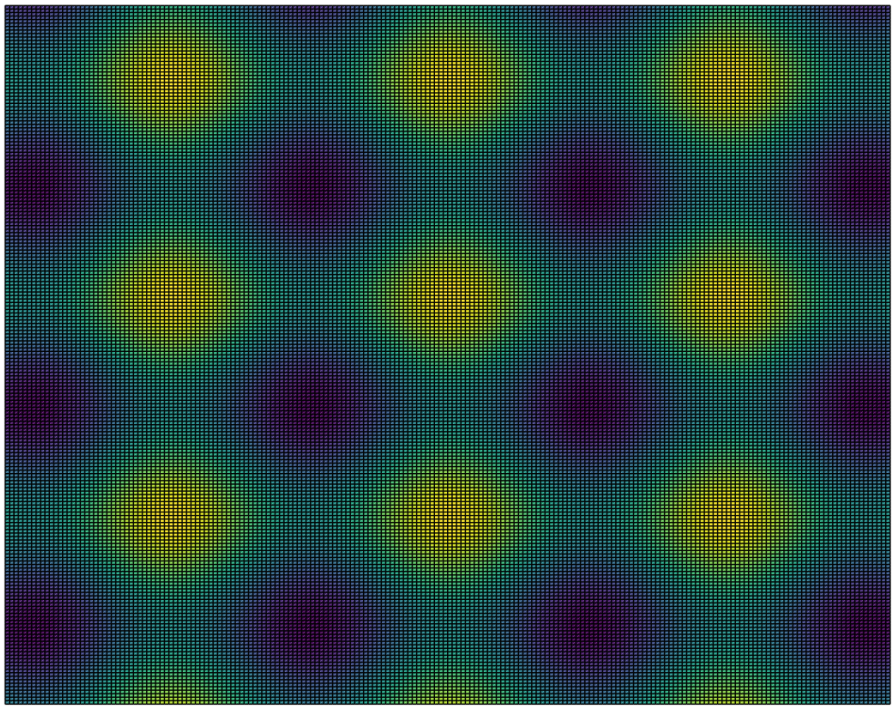
Gaussian distribution is very often used continuous probability distribution. The Gaussian distribution is significant in statistics and is commonly used in the natural and social sciences to represent real-valued random variables. The probability density function formula for multivariate Gaussian distribution is given by:

$$p(\mathbf{x}; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(x-\mu)^T \mathbf{\Sigma}^{-1}(x-\mu)\right) \tag{2}$$

where vector $x = (X_1, X_2, \ldots, X_n)$ is the linear combination of random variables $X_1, X_2, \ldots, X_n$ and $\mathbf{x} \sim \mathcal{N}(\mu,\ \Sigma)$, μ is the mean vector and Σ is the covariance matrix. This experiments aims to show if the network can adjust its detection pattern according to the point density modeled as a multivariate Gaussian distribution, hence the feature patterns, see Figure 19.

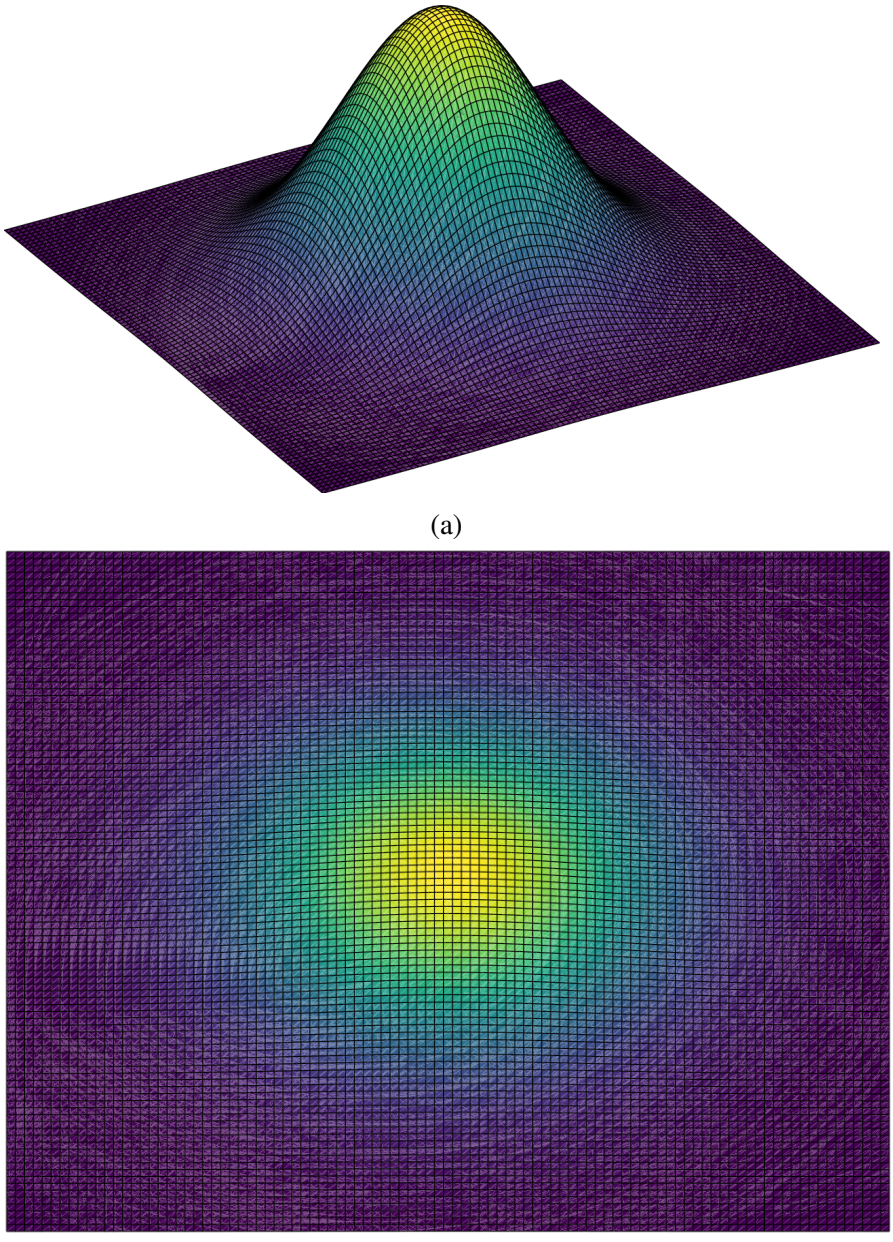


(a)



(b)

Figure 18. (a) Sinusoidal function representation in 3D (b) Bird's-eye-view representation of bumps in feature maps.



(a)

(b)

Figure 19. (a) Gaussian function representation in 3D (b) Bird's-eye-view representation of Gaussian in feature maps.

3.2.4 Noise Encoding

Uniform distribution can be one of the most straight-forward distribution functions to keep in mind. The probability distribution function of the continuous uniform distribution is:

$$f_{XY}(x, y) = \frac{1}{A} \tag{3}$$

where *A* is the area of the square or the circle. Since the definition of a "uniform distribution" is that the density function is constant for all x,y within the defined boundary region, this formula should apply to Cartesian coordinates as well. In feature map case, width and height can be thought of as the defined boundaries.

This experiment aims to show that noise encoding does not work better than true positional encoding schemes.

3.3 360-degree Detection

Recent advances in deep learning models have been extensively addressed to the quantity and diversity of data collected in recent years. Data augmentation is a handy tool that lets us considerably increase the variety of data available during training, without actually gathering and labelling new data. For instance, when it comes to the camera modality, well-known and standard data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train more robust and capable neural networks. In this work, data augmentation techniques such as rotation and translation of points globally and ground truth level are used as the data source consists of lidar point cloud.

In this work, ground truths exist only in camera FOV, which reveals the need to apply data augmentation during training or inference to have 360-degree detections. During training lidar point clouds are augmented replicating FOV three times while during inference it is managed rotating the point cloud and running the network on FOV three times.

3.3.1 FOV Rotations During Inference

Another way of having 360-degree detections is to rotate the FOV three times during at inference time. Since the model is trained on the point cloud in FOV, it is inclined to detect objects in FOV. One of the reason it quickly fails to detect objects outside of FOV is that the data distribution in FOV and the outside of FOV are different. To alleviate this issue, at least during inference, the following steps are applied:

- Run the network on FOV and get detections.
- Rotate the point cloud 90-degrees clockwise and run the network on the new FOV. Here assuming that camera FOV is 90-degrees. Otherwise, the number of rotations around z-axis should be adjusted accordingly.

- Get detections and rotate them back to their correct positions.

This process is applied for three other sectors that are outside of FOV. In the end, the model will have been run four times. Therefore, it is expected that the run-time of the network will be slower.

3.3.2 FOV Replications During Training

Ground truths from FOV are copied and transformed properly to the other three sectors in the point cloud. Therefore, before starting a training network on 360-degree view, a several steps of processing needs to be realized, also see Figure 21d:

- Transforming ground truths from camera coordinate frame to lidar coordinate frame.
- Taking the FOV region of the point cloud.
- Transform the FOV and ground truths three times by 90-degree around z-axis, assuming camera FOV is 90-degrees. Otherwise, the number of transforms around z-axis should be adjusted accordingly.
- Concatenate the original FOV with three transformed version of it.

3.3.2.1 Field-of-View Replication - 360 degree After successfully completing the preprocessing, it is recommended to visualize the point cloud with the all ground truths so that the model that will be trained does not get confused with wrongly positioned ground truths, see Figure 21d.

In this experiment, only Milrem dataset is used for trying the data augmentation technique. The reasons for this decision are the following:

- KITTI dataset has 64-beam lidar data while Milrem dataset has 16-beam lidar data.
- The eventual model will be used in an environment similar to one in which Milrem UGVs operate.

3.3.2.2 Field-of-View Replication - 180 degree After the model trained on 360-degree view gives poor results, see Figure 20, than expected, another experiment is made replicating FOV only once. In the end, point cloud consists of points from FOV and the rotated version of FOV along with ground truths adjusted, see Figure 21c. In this experiment, too, only Milrem dataset is used for trying the data augmentation technique.

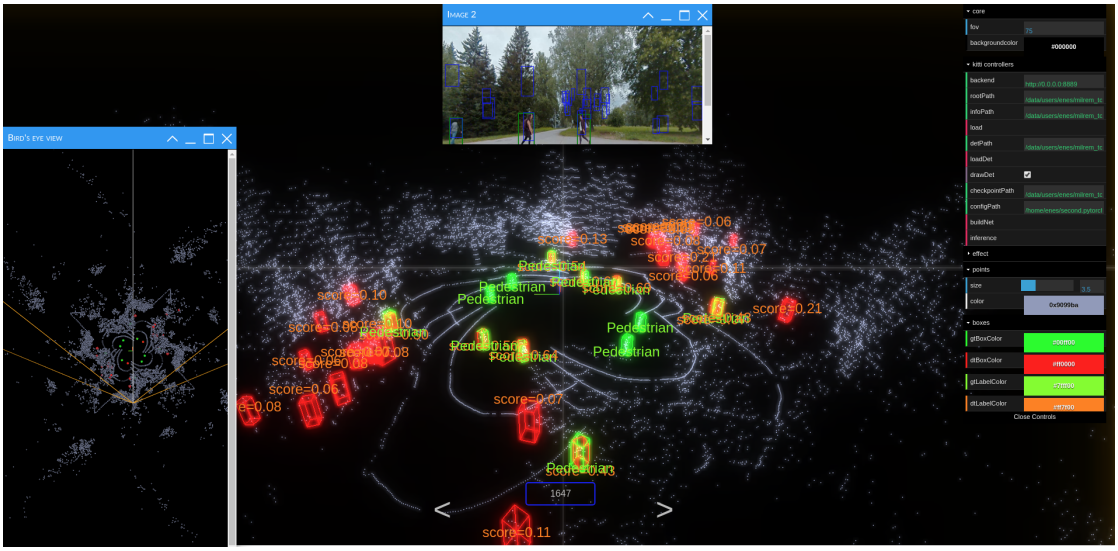
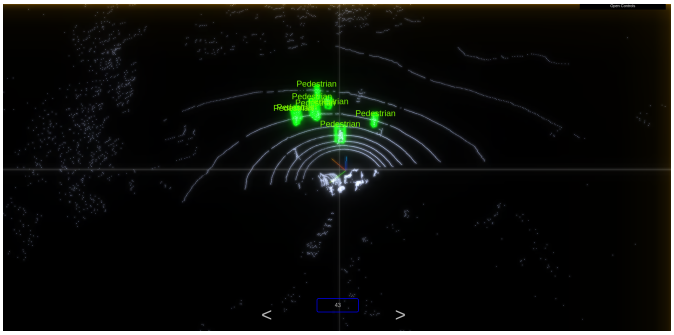
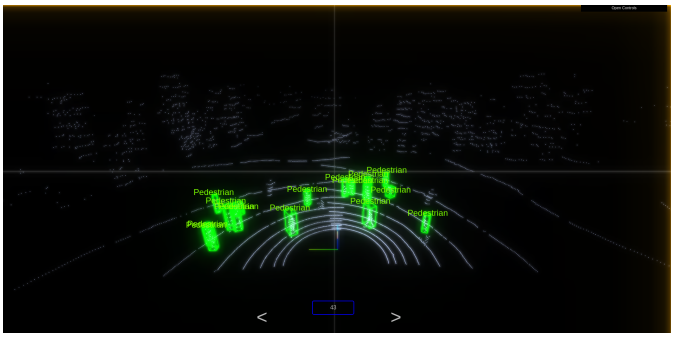


Figure 20. Green colored boxes are ground truths. Red colored boxes are False-Positives. Orange colored boxes are True-Positives. The left pane shows the BEV of the point cloud along with the boxes. The right pane is used for adjusting the settings of the visualization.

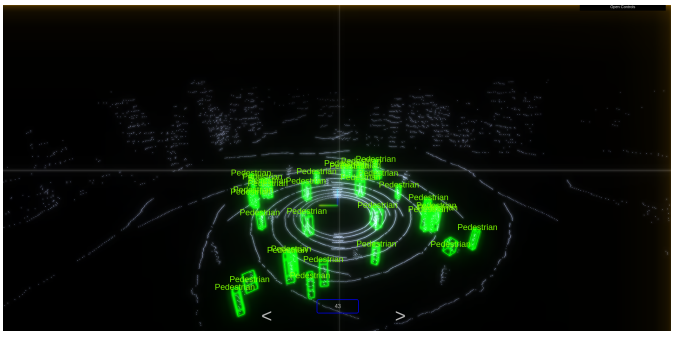
(a)



(b)



(c)



(d)

Figure 21. (a) Camera image with lidar ground truths projected onto. (b) Original FOV ground truths are visualized. (c) Point cloud augmented for 180-degree view with lidar ground truths. Original point cloud and ground truth are duplicated and rotated 90 degrees clockwise. (d) Point cloud augmented for 360-degree view with lidar ground truths. Original point cloud and ground truth are duplicated 3 times and rotated 90, 180 and 270 degrees clockwise. All 2D & 3D ground truths are shown with green boxes and cuboids.

4 Results

This chapter shows the results obtained on the test datasets of the respective datasets in each experiment.

4.1 KITTI

On KITTI dataset, a baseline model and another model with linear encoding scheme are trained for only car class. Those results are used to infer if CoordConv is useful at all. According to the official KITTI evaluation, improvements are consistent with respect to all of the metrics.

The first table, see Table 1, shows the numerical results for two models (Baseline and CoordConv-Linear) on KITTI Car dataset. For the evaluation, IoU threshold (0.7) is the same for all the difficulty levels. CoordConv model outperforms the baseline model with respect to each metric, besides, this consistent increase in the performance is valid for the other IoU thresholds (0.7:0.5:0.5), see Table 2. Each IoU threshold applies to different difficulty levels easy, medium and hard consecutively. The reason for this is that objects in hard category are more difficult to detect than objects in easy category.

Car(AP=(0.7:0.7:0.7))				
Model	**Metric**	**Easy**	**Moderate**	**Hard**
Baseline	bbox	90.64	88.77	87.23
Baseline + Linear*	bbox	90.75	89.16	87.83
Baseline	bev	89.86	86.44	80.93
Baseline + Linear*	bev	90.02	86.97	84.67
Baseline	3d	80.38	75.11	69.61
Baseline + Linear*	3d	82.69	76.04	69.77

Table 1. Results on the KITTI dataset with the baseline model and CoordConv model.
*: Linear encoding scheme outperforms the baseline model in all of the metrics.

4.2 Milrem

After getting the intuition that CoordConv helps the network achieve better results according the experiments made on KITTI dataset, models with different encoding schemes are trained on the Milrem dataset for Pedestrian class.

Car(AP=(0.7:0.5:0.5))				
Model	**Metric**	**Easy**	**Moderate**	**Hard**
Baseline	bbox	90.64	88.77	90.64
Baseline + Linear*	bbox	90.75	89.16	87.83
Baseline	bev	90.78	89.92	89.31
Baseline + Linear*	bev	90.82	90.12	89.38
Baseline	3d	90.77	89.79	89.08
Baseline + Linear*	3d	90.82	89.94	89.11

Table 2. Results on the KITTI dataset with the baseline model and CoordConv model.
*: Linear encoding scheme outperforms the baseline model in all of the metrics.

Pedestrian(AP=(0.5:0.5:0.5)) & BBOX metric			
Model	**Easy**	**Moderate**	**Hard**
Baseline	0.02	0.02	0.02
Baseline + Linear	26.13	26.13	26.13
Baseline + Gaussian	23.59	23.47	23.47
Baseline + Sinusoidal	19.73	19.70	19.70
Baseline + Noise*	29.42	28.63	28.63

Table 3. Results using 'BBOX' metric on the Milrem dataset with the baseline model and all types of CoordConv models. Baseline bbox results are quite poor due to the misalignment of 3D detections in z-axis, meaning that their projection to the image is not well aligned. In KITTI, configuration matrices provided are more reliable.
*: Noise encoding scheme outperforms all others.

Noise encoding surprisingly outperforms all others, see Table 3. The reason for this can be that it helps the network generalize better, discussed further in section 5.

Pedestrian(AP=(0.5:0.25:0.25)) & BEV metric			
Model	**Easy**	**Moderate**	**Hard**
Baseline	81.23	81.12	81.12
Baseline + Linear	84.31	84.24	84.24
Baseline + Gaussian*	85.69	85.61	85.61
Baseline + Sinusoidal	84.38	84.25	84.25
Baseline + Noise	83.62	83.71	83.71

Table 4. Results using 'BEV' metric on the Milrem dataset with the baseline model and all types of CoordConv models.
*: Gaussian encoding scheme outperforms all others.

All of the models achieve good results, yet, Gaussian model still outperforms others, see Table 4.

Pedestrian(AP=(0.5:0.25:0.25)) & 3D metric			
Model	**Easy**	**Moderate**	**Hard**
Baseline	76.95	76.74	76.74
Baseline + Linear	80.89	80.82	80.82
Baseline + Gaussian	81.67	81.61	81.61
Baseline + Sinusoidal*	81.87	81.82	81.82
Baseline + Noise	80.67	80.47	80.47

Table 5. Results using '3D' metric on the Milrem dataset with the baseline model and all types of CoordConv models.
*: Sinusoidal encoding scheme outperforms all others.

All of the models achieve better results, yet, Sinusoidal model outperforms others, see Table 5.

4.3 360-degree Detection

360-degree detection, unlike the other models, do not perform well. From Figure 20, it can be seen that the reason for 360-degrees model to give poor results is that there are too many False-Positives, see Table 6.

Pedestrian(AP=(0.5:0.25:0.25)) & 3D metric			
Model	**Easy**	**Moderate**	**Hard**
Baseline	76.95	76.74	76.74
Baseline + 360	37.28	37.16	37.16

Table 6. Results using '3D' metric on the Milrem dataset with the baseline model and the 360-degrees model

4.4 Runtime of the Models

This sections shows the runtime of the models run on one of Milrem datasets. The results demonstrates that 360-degree detection method does not degrade the speed too much. Furthermore, they prove that CoordConv models run at a similar speed as the baseline model, see Table 7.

Runtime Comparison	
Model	**FPS(Frames per second)**
Baseline	11.94
CoordConv	11.17
Baseline + 360	11.46
CoordConv + 360	10.91
Baseline + 360(rotation)	3.08
CoordConv + 360(rotation)	2.89
Baseline + 360(mirror)	6.36
CoordConv + 360(mirror)	5.68

Table 7. Results showing the runtime of the models on a Dell laptop*.
*: Graphics: GeForce GTX 1050 Ti with Max-Q Design/PCIe/SSE2 4GB

There are three different versions of run-times measured for 360-degrees detection. First one is to pass the network the entire pointcloud and get detections. This version provides the best inference timing among the other 360-degrees models. The second approach is to rotate the pointcloud and run the model on FOV four times in total. It is expected that sequential inference takes the longest time to get detections. The third option is to run the model separately on two halves of the pointcloud. The model runs on the first half(area in front of the UGV) of the pointcloud, then, runs on the mirrored version of the pointcloud. By mirroring, points behind the UGV are repositioned to the front of the UGV. It works faster than the rotation method as it handles the entire pointcloud in two steps.

4.5 Summary

CoordConv models provide the existing network with an improvement in accuracy without sacrificing too much speed, see Table 7, as the number of parameters introduced with CoordConv models are not many. On the other hand, 360-degrees models show some differences in terms of both accuracy and speed. For instance, running the model on the entire pointcloud at once gives comparable performance as the FOV models (Baseline and CoordConv). However, as shown in Figure 20, the number of False-Positives are quite high due to the reason that the model is trained on FOV and the data distribution in pointcloud varies from one sector to another. Rotation and the mirror methods are also tried for 360-degrees experiments. They are better than the first approach in terms of accuracy, yet, suffering from being too slow. To sum up, CoordConv models are good enough to replace the Baseline model as it improves accuracy while not degrading the performance too much. Coming to the 360-degree models, running the model on the entire pointcloud is the option that needs more attention since it works comparable faster than other 360-degrees approaches (rotation and mirror). Therefore, in the end, a CoordConv model trained and working reasonably well on the entire pointcloud, not only in FOV, should be the desired approach.

5 Discussion

Noise encoding scheme unexpectedly improves the results as demonstrated in Table 3. This section aims to shed light on possible reasons why noise is useful.

In Machine Learning world, overfitting is a phenomena which is resulted from the model relying too much on training data, and therefore, not generalizing well on unseen data. We always want our models to generalize and perform reasonably well on unseen data, otherwise, they can easily fail achieving the tasks they are supposed to manage.

One of the ways of regularizing the model so that it does not overfit the training data is to add random noise to the input during training. This effectively enables the model to learn a general mapping from input space to output space, or else, the model might tend to memorise the specific inputs and their associated outputs. In other words, noise injection can act as a data augmentation technique [27].

Another advantageous that noise injection provides is that it indeed synthetically creates new data apart from the original dataset. More training data allows the model to be exposed to a richer information of the problem, this way making the model learn the more important features from the input.

Normally, one would expect random noise to make the model perform poorly. However, in practice, it has been showed that noise can help the model achieve better generalization [25].

Adding random noise also provides the model with a regularization effect, hence, enhance the robustness of the model [26].

So far, it has been discussed in which way adding noise to the existing input values can help the model. However, in this work, noise is introduced as a new input or channel that the model should learn to ignore. In other words, CoordConv model is supposed to assign zeros to all weights corresponding the noise channels.

Random noise uniformly distributed is added to the input before each convolutional and deconvolutional layer in one of the experiments, see subsubsection 3.2.4 and Table 3. In that experiment, noise encoding might behave as a regularizer and help the model generalize better, besides, improve the robustness of the model. In other words, it might help the network to figure out the more general features from the sample point clouds.

To sum up, even though the aim of usage of the noise in this thesis is to show that noise encoding does not improve the accuracy as CoordConv is supposed to ignore useless positional channels, noise helps the network achieve good results in experiments. Considering the practical usefulness of the regularizers in Neural Networks (NNs), noise encoding needs a deeper look to exploit the benefits from it. For instance, visualizing the activation of the filters on noise channels might be an idea to see how CoordConv layers behaves on those channels.

6 Conclusion

This work shows that Positional Encoding in Convolutional Layers in Region Proposal Network of the main architecture of PointPillars improves detection accuracy. Positional encoding is just a helper to tell the CNN filters where they are currently on the input as they are translation invariant by nature. As maintained in the work [9] done by Uber AI Labs, the network is supposed to make use of positional channels if those channels are helpful.

Quantitative results show that they are helpful to some extent. The last experiment, using noise encoding scheme, is to encode noise channels sampled uniformly between -1 and 1. Its better performance than the baseline model could be due that it helps the model generalize better.

All of the CoordConv models are trained for only camera Field-of-View, besides, 360-degree model does not work as expected. Therefore, as a future work, the main point of interest should be to get a model working on 360-view reasonably. Another potential are of the work is to try out different data augmentation techniques [24] recently offered by Waymo.

References

[1] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In ICRA, 2017

[2] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In CVPR, 2017

[3] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander. Joint 3d proposal generation and object detection from view aggregation. In IROS, 2018.

[4] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In CVPR, 2018

[5] B. Yang, W. Luo, and R. Urtasun. PIXOR: Real-time 3d object detection from point clouds. In CVPR, 2018.

[6] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, Oscar Beijbom. PointPillars: Fast Encoders for Object Detection from Point Clouds. In CVPR, 2019.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y.Fu, and A. C. Berg. SSD: Single shot multibox detector. In ECCV, 2016.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. In NIPS, 2017.

[9] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, Jason Yosinski. An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution. In NeurIPS, 2018.

[10] Andreas Geiger, Philip Lenz, Christoph Stiller and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. In I. J. Robotics Res, 2013

[11] Y. Yan, Y. Mao, and B. Li. SECOND: Sparsely embedded convolutional detection. Sensors, 18(10), 2018.

[12] M. Simon, S. Milz, K. Amende, and H.-M. Gross. ComplexYOLO: Real-time 3d object detection on point clouds. arXiv:1803.06199, 2018.

[13] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. In CVPR, 2018.

[14] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander. Joint 3d proposal generation and object detection from view aggregation. In IROS, 2018.

[15] Charles R. Qi, Hao Su, Kaichun Mo, Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In CVPR, 2017.

[16] Convolutional Neural Networks. https://cs231n.github.io/convolutional-networks/

[17] Convolutional Neural Networks. https://en.wikipedia.org/wiki/Convolutional_neural_network

[18] Alex Krizhevsky and Sutskever, Ilya and Hinton, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks.In NIPS, 2012.

[19] Lidar. https://velodynelidar.com/what-is-lidar/.

[20] PointCloud https://mc.ai/lidar-sensor-modeling-and-data-augmentation-with-cyclegar

[21] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. International Journal of Computer Vision, 2010.

[22] Transformer explained. https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.

[23] The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Development Kit.

[24] Shuyang Cheng and Zhaoqi Leng and Ekin Dogus Cubuk and Barret Zoph and Chunyan Bai and Jiquan Ngiam and Yang Song and Benjamin Caine and Vijay Vasudevan and Congcong Li and Quoc V. Le and Jonathon Shlens and Dragomir Anguelov. Improving 3D Object Detection through Progressive Population Based Augmentation. arXiv:2004.00831, 2020.

[25] Train Neural Networks With Noise to Reduce Overfitting. https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/.

[26] Chris M. Bishop. Training with Noise is Equivalent to Tikhonov Regularization. Neural Computation, 1995.

[27] Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning (Adaptive Computation and Machine Learning series), 2016.

[28] AlexNet architecture. https://livebook.manning.com/book/grokking-deep-learning-for-computer-vision/chapter-5/v-8/65.

[29] S. Mohamed, Ihab. (2017). Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques. 10.13140/RG.2.2.30795.69926.

[30] Velodyne point cloud. https://www.cnet.com/roadshow/news/how-lasers-map-the-world-for-self-driving-cars/.

[31] Joseph Redmon and Ali Farhadi.YOLOv3: An Incremental Improvement. arXiv:1804.02767, 2018.

[32] Transformer. https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04.

[33] Precision and Recall. https://en.wikipedia.org/wiki/Precision_and_recall.

[34] Precision-Recall curve. https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3.

[35] Intersection over Union. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[36] Sinusoidal Frequencies. https://www.radartutorial.eu/10.processing/sp53.en.html.

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Enes Ozipek**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Real-time 3D Object Detection on PointClouds,
   *(*title of thesis)

   supervised by Tambet Matiisen.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Enes Ozipek
15/05/2020