Seeing Autonomous Vehicles: Applying Deep Learning to LIDAR Point Cloud Data

**Authors: Talha Muhammad**

ABSTRACT: Autonomous vehicles face a variety of challenges – including accurate object detection (i.e. distinguishing cars from pedestrians) and more importantly 3D and depth perception of objects. Data input from Light Detection and Ranging (LIDAR) sensors can be processed to develop machine learning models, which can in turn be used to more accurately and rapidly deploy autonomous vehicles. In 2019 Lyft released its Perception dataset which includes camera and LIDAR data collected from vehicle driving scenes. The data are annotated with bounding box projections and object classifications.

In this paper we develop and apply different neural network architectures for training and testing / validation to compare object detection and perception performance by utilizing the Lyft dataset and the associated Software Development Kit (SDK). We find that a number of model hyper parameters can have a significant impact on the model results, in particular voxelization of the LIDAR point cloud can impact the model results. Finer voxelization of the LIDAR point cloud leads to better model performance.

The approach in this paper is to apply the existing state of the art (rather than developing new state of the art methods) to the Lyft dataset. Another goal of the paper is to utilize and process LIDAR data for data visualization, model development for object detection and 3D bounding box projection. We apply a variety of network learning architectures to LIDAR bird’s eye-view (BEV) representations including different Unet architectures.

# Introduction

The transportation sector is currently undergoing significant changes and developments, particularly in the past five years with the coming online of mobile apps and the development of technology platforms like Uber, Lyft and others. With the advent of mobile apps together with great advancement in machine learning including computer vision together with computing power and advancement have all encouraged the development, research and testing of autonomous vehicles.

The ongoing development of autonomous vehicles is changing the transportation landscape rapidly – with even greater transformation expected in the future. Autonomous vehicles pose a variety of challenges – most importantly accurately identifying objects / cars on the roadway using inputs from sensor data. Data input from sensors like LIDAR can be processed to develop machine learning models, which can in turn be used to more accurately and rapidly deploy autonomous vehicles. In 2019 Lyft released its Perception dataset which includes camera and LIDAR data collected from vehicle driving scenes. The data are annotated with bounding box projections and object classifications. A major challenge that autonomous vehicles face is accurate object detection (i.e. distinguishing cars from pedestrians) and more importantly 3D perception and the depth of objects. In this paper we develop and apply different neural network architectures for training and testing / validation to compare object detection and perception performance by utilizing the Lyft dataset and the associated Software Development Kit (SDK). We find that a number of model hyper parameters can have a significant impact on the model results, in particular voxelization of the LIDAR point cloud can impact the model results. Finer voxelization of the LIDAR point cloud leads to better model performance.

# Literature Review

Convolutional Neural Networks (CNN) have become an increasingly important tool in computer vision problems and have evolved rapidly in past few years in developing different structures to efficiently solve these problems. The section below is organized as follows, 2.1 provides a technical overview including the introduction of different concepts that are relevant for the analytical methods used, 2.2 provides a review of different network architectures that have been used in CNN’s, 2.3 provides a review of different approaches that have been adopted for conducting 3D object detection and processing Point Cloud (or LiDAR) data and 2.4 provides an overview of the different datasets that have been used in the literature to develop models to solve this problem.

## 2.1 Technical Overview

Typical CNN’s have neurons arranged in 3 dimensions of width, height and depth as input, with the depth representing 3 color channels (RGB) for images. Compared to a typical neural network which consists of multiple layers of fully connected hidden layers, a CNN consists of a sequence of layers that convolve the input and yield an output i.e. convert the input to an output after performing a differentiable function on the input [1].

CNN’s have fewer or in some cases no fully connected layers and thus greatly reduce the number of parameters required to be estimated and learned during model training process. A simple CNN may consist of a convolution layer, a pooling layer and fully connected layer. The convolution layer typically has different filters which are applied to the input to extract different features of the output. For example, one filter may extract only vertical lines in the input, and another may extract only horizontal lines from the input. The number of filters associated with a convolutional layer and the size (width and height of the filter kernel) are hyperparameters for the convolutional network. The specific weights of the filter kernel are learned through training by minimizing a loss function similar to the case of fully connected layers [1]. Max pooling layers are used to reduce the size of the input generally referred to as down-sampling, with the idea being to identify key features of the input by reducing the size of the input.

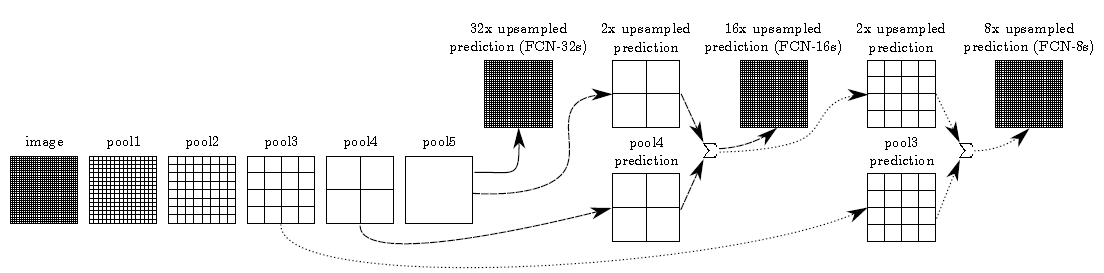
Deconvolution layers, also referred to as transposed convolution layers, take an input and produce an output that that is of greater size and are generally incorporated in a number more recent network architecture. Deconvolutional layers can be used as a decoding layer for a convolutional layer or to project feature maps to a higher dimensional space [2]. Deconvolution layers operate in a top-down fashion generating the input signal by combining convolutions of the feature maps with learned filters [3]. The approach allows for the unsupervised construction of hierarchical image representations. For example, in [6] the authors show that deconvolutional networks can learn different features which can be combined in the reconstructed image with object detection. Deconvolution layers are used extensively in the literature as part of the network architecture to extract higher level features from point cloud and image-based datasets.

Fully convolutional networks (FCN) are another adaptation that employ convolutional networks for object detection and semantic segmentation [4,5]. FCN’s are unique in that they consist only of convolutional layers – convolution, pooling and up-sampling - and do not contain any fully connected hidden neural layers. This greatly reduces the number of parameters needed for computation of the network and thus are much faster than the typical convolutional network which has fully connected layers. In addition to this, a key insight of FCN’s is that any fully connected network can be converted to its corresponding FCN. The densely connected layer can be considered as a 1x1 convolutional layer [4]. In [4] the authors develop FCN versions of a number of convolutional networks such as LeNet, AlexNet and VGGNet.

FCN’s have two different path’s a down-sampling path (which reduces the size of the output during convolution) and an up-sampling path (increases the size of the output during convolution). The down-sampling path recovers local semantic and contextual information (the what) while the up-sampling path recovers the spatial information (the where). Up-sampling paths are deconvolutional layers while down-sampling paths represent typical convolutional layers.

To recover and combine the information learned through each path, skip connections are used. Skip connections consist of connections that link the output of one layer to the input of another but skip or exclude in between layers so as to incorporate outputs of different layers concurrently. The figure below shows an example [as develop in 4]. FCN 32 is the output of a 5th layer of pooling (and convolution – convolution layers are not shown for brevity). FCN 16 is obtained by concatenating output from the 5th layer of pooling and a (skipped) 4th layer of pooling.

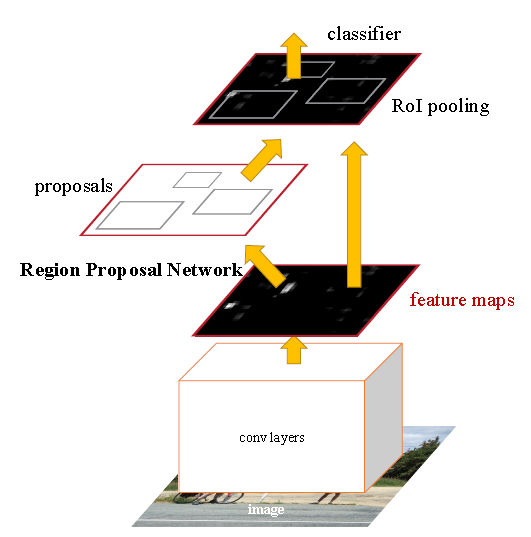
Figure 1: Skip connections for Semantic Segmentation

****

Source: [4]

RPN (Regional Proposal Networks) is a network that takes an image input of any size and simultaneously outputs rectangular object proposals together with a particular score for the object [17]. In [17] the authors introduce a novel RPN that shares convolution layers with an object detection network and thereby reducing time and cost for computing proposals. In simple words, the RPN network tells the object detection network *where* to look in a particular image. The authors thus propose a two-stage approach, with a first stage of proposal generation followed by object identification in the second stage as compared to other approaches that conduct single stage identification of region and object detection. Based on the analysis, the two-stage approach has superior performance relative to the one stage approach. By using novel anchor boxes that use references at multiple scales and aspect ratios, the authors in [17], improve performance of the network (as shown in the figure below). The RPN is combined with the object detection network by a training scheme that alternates between loss reduction for the region proposal task and loss reduction for the object detection task, resulting in a unified network with convolutional features that are shared by the both tasks.

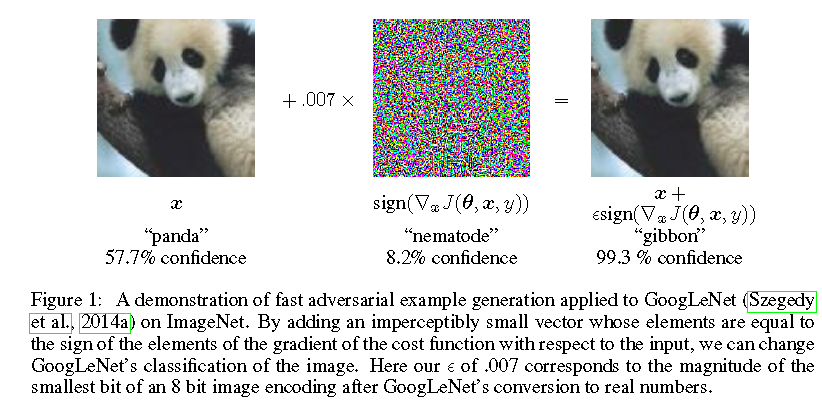
Figure 2: An RPN combined with a R-CNN



Source: [17]

One challenge with neural networks that have been documented in the literature is that of adversarial examples [7]. Adversarial examples occur when a small perturbation in the dataset results in neural networks misclassifying the output with a high degree of confidence. For example, in some cases such as ImageNet, the adversarial examples were so close to the original examples that the differences are indistinguishable to the human eye. The authors recommend training with adversarial examples so that models are more regularized.

Figure 3: An Adversarial Example applied to GoogLeNet on ImageNet



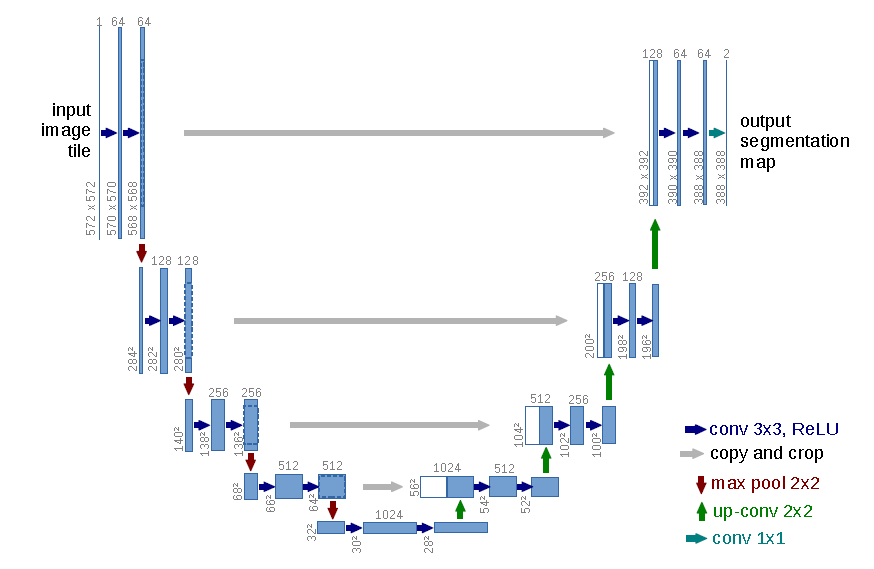
Source: [7]

## Network Architecture

The current state of the art references and uses a number of architectures. One of the main training architectures that are being used is the Unet [9]. The approach builds upon the FCN architecture that was discussed above. In [9], the authors adopt the FCN approach with some modifications that works with very few training images and provides precise segmentations. This approach won the 2015 International Symposium on Biomedical Imaging (ISBI). The figure below (taken from the paper) illustrates the approach and how it works. As can be seen in the figure the architecture consists of a contracting / down-sampling path and an expansive / up-sampling path. The down-sampling path consists of two convolutions followed by ReLU layers and a max-pooling layer. After each max pooling layer during the down-sampling phase the number of feature maps are doubled just as the size of output is down-sampled by a factor of two.

During the up-sampling phase, at each the level there are two convolutional layers followed by an up-convolution (synonymous with the deconvolutional layers we discussed above). The number of features are reduced by a factor of two just as the size of the output is increased (see Figure 4 below). A process of concatenation takes places that combines the up-sampled feature from the corresponding down-sampled features. The benefit of this concatenation is that during the down-sampling process very localized features are captured while during the up-sampling process higher level features are captured. At the final layer a 1x1 convolution is used to map the 64 component features vector to the desired number of classes.

Figure 4: The Unet Architecture



Source: [9]

Sparse CNN are another approach used to improve performance of convolutional networks and indeed to train deeper networks. In [10] the authors develop an approach to more efficiently represent data in sparse form. For forward propagation the authors calculate two matrices for each layer of the network, which consists of a feature matrix and a pointer matrix.

The feature matrix is a list of row vectors one for the base state (background for example) and one for the active spatial location in the layer. The number of features per spatial location is represented by width of the matrix. The pointer matrix is of size equal to the convolution layer. For each spatial location in the convolution layer, the number of corresponding in the feature matrix is stored.

The benefit of using sparse CNN’s is that they allow the training of faster networks and allow for the training of much deeper networks with comparable performance.

## Applications to 3D Object Detection and Point Cloud

This section presents approaches that have been covered in the sections above from a theoretical and application perspective but now specifically applied to applications for data that are applicable for autonomous vehicles. Two types of dataset are useful and applicable for autonomous vehicles that will be considered in this paper, namely image data and LIDAR (or point cloud data). There are other sources of data which will be briefly discussed but will not be considered for this analysis.

Image data provides good feature resolution and object detection capabilities however depth detection which is critical for 3D object detection and for applications in autonomous vehicles is not well represented. LIDAR data on the other hand provides good depth detection however it does have its own challenges. In particular, data from LIDAR is represented as a cloud of points (hence the name point cloud) with each frame having 100k or more points. This means the point locations need to be processed and can take up computational resources. In addition, approach speed of the vehicle that is hosting the LIDAR equipment can also impact the number of points per object available for processing. Finally, LIDAR points are sparse, have highly variable point density due to factors such as non-uniform sampling of the 3D space, effective range of sensors along with other factors and therefore require additional considerations.

Combining both LIDAR and image data together for processing may present a solution however the approach comes with additional challenges. For one, image capture cameras and LIDAR sweeps can become unsynchronized and so some additional processing is needed to ensure both sources are synchronized. Secondly, with such an approach, much more data needs to be processed which can slowdown the network and image segmentation / classification.

In the literature therefore four main approaches are generally used to deal with 3D object detection based using point cloud and image-based data [16].

The first consists of front view (camera) or image-based methods. These are generally based on 2D image representations of image data or Bird’s eye view (BEV). In front view methods, image-based methods are used to generate 2D bounding boxes including in some approaches utilizing CNN’s for analysis. In some approaches for analyzing front view LIDAR data are also used [14]. Generally, these methods do not perform well for 3D detection [16].

A second approach consists of first converting point cloud data into a BEV representation. In this case point cloud data are used to obtain height maps which are then combined with point intensity and density maps to obtain features [12]. However, one problem with these approaches is that there is a loss of information when developing the BEV map. This reduces performance of these models particularly for 3D bounding box applications.

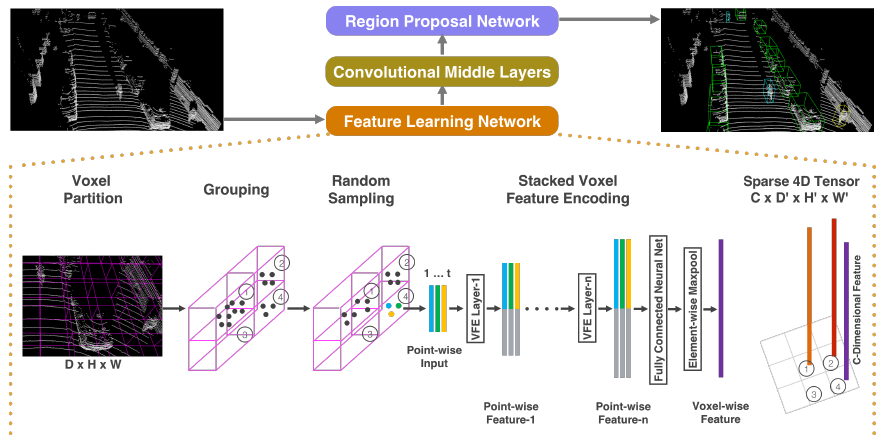
A third approach considers methods and applications in 3D. Methods in 3D generally work with LIDAR data (although there are other data sources such as stereo images) by converting point cloud data into 3D grids or voxels. In [13], the authors exploit the sparse nature of 3D data to develop a voting scheme that is equivalent to a convolution on a sparse feature grid. Using point cloud data they extract six features which include three measures based on the diffusion of the point cloud (these measures consider eigenvalues of the point cloud in 3D space, the mean and the variance of the reflectance values of the points and binary occupancy feature.

In [11] a generic 3D detection network is developed that conducts feature extraction as well as bounding box prediction into a single stage deep network. Instead of manual feature engineering this approach uses machine learned features. The authors divide the point cloud into equally spaced 3D voxels and transform a group of points within each voxel into a unified feature representation through a voxel feature encoding layer (VFE). The VFE layer then feeds into a RPN to generate detections. RPN’s are highly optimized for object detection however it requires data to be dense and organized in a tensor structure – which not typically the case for LIDAR which tends to be sparse.

As shown in the figure below the approach consists of 3 different steps: 1) feature learning network 2) convolutional middle layers and 3) regional proposal network (see Figure 5 below).

The feature learning network itself consists of a number of steps, these include as a first step creation of the voxel partitioning (essentially creating partitions of 3D space and grouping points according the partitioned space). Secondly, random sampling of LIDAR points from the partitioned voxel space. LIDAR is high definition consisting of ~100k points with highly variable point density. Random sampling provides computational savings and reduces the imbalance of points due to the variable point density.

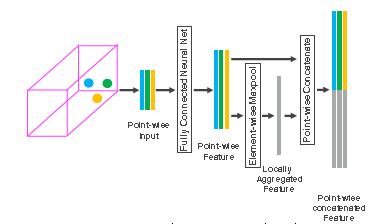
Figure 5: VoxelNET Approach

****

Source: [11]

Thirdly, stacked voxel feature encoding implements skip connections and concatenation to learning pointwise and locally aggregated features (see Figure 6 below). Fourthly, while LIDAR points consist of 100K points per LIDAR scene, about 90% of the voxels are empty, and the sparse tensor only considers non-empty voxels. Thus the sparse tensor representation greatly reduces memory usage and computation cost.

Figure 6: Stacked VFE Layers

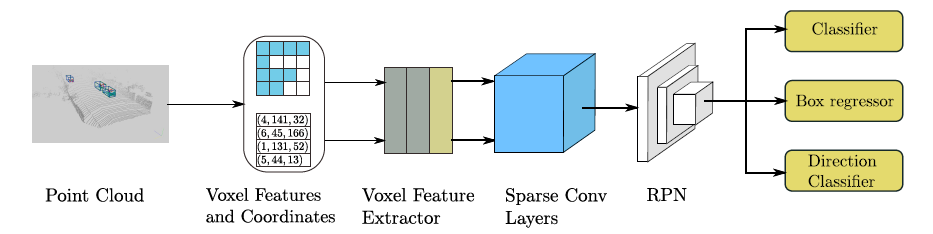


Source: [11]

The RPN in [11] consists of a number of convolutional layers that successively down-sampled the layers. Successive deconvolutional layers together with skip connections and deconvolutional layers up-sample the output which is concatenated and outputs a location and probability score map.

In [16] the authors only use LIDAR data to convert point cloud data into voxels and then applying linear networks eventually converting these into 3D tensors which are ultimately used in a region proposal network. The authors use a similar structure as [11] above with some notable changes including sparse convolution methods which greatly improve speed and training of these networks. In addition, in [16] the authors use an angle loss regression framework to improve orientation performance and data augmentation approaches to enhance convergence speed and performance.

Figure 7: SECOND Detector



Source: [16]

In [15] the LIDAR data is used on the Nuscenes dataset, using sparse 3D convolution to extract features which are then fed into a multi-task learning framework. The authors develop and apply a method to conduct multi-task learning by estimating and training all categories together rather training them individually.

Finally, a fourth approach considers a fusion based approach that combines both camera images with point cloud data. In [12] point cloud data are used to develop both front view feature maps and BEV based feature maps. These point cloud maps are then combined with an image feature map. The network performs better than the BEV only network however, performance is slower due to the model containing three CNN’s.

## Object Detection and Point Cloud Datasets

A number of different datasets are available the provide image and point cloud data for model training, testing and validation. The most well documented and extensive studied and used is the KITTI dataset [18]. The KITTI 3D object detection database consists of 7,481 training images / points clouds and 7,518 test images / point clouds covering three categories: car, pedestrian and cyclist. For each class detection, three difficultly levels are available for evaluation – easy, medium and hard. The categorization is based on object size, occlusion and truncation level [11]. For KITTI 3D object detection is based on LIDAR. On other hand Cityscapes [20] has released a dataset together with 3D bounding boxes with only RGB / stereo images (and not using LIDAR for depth perception).

Recently Nutonomy (the nuScenes dataset) released autonomous vehicle model training dataset and carries a full autonomous vehicle sensor suite, consisting of 6 cameras, 5 radars and 1 lidar providing 360 degree field of view [21]. NuScenes comprises of 1000 scenes (each 20 seconds long), fully annotated 3D bounding boxes for 23 classes and 8 attributes. In [21] devkit, evaluation code, taxonomy and database schema are provided. NuScenes is a much more challenging dataset than KITTI, for one it requires detection of 10 categories simultaneously compared to 3 in the KITTI dataset. Additionally object classes in the dataset are significantly imbalanced making the nuScenes more challenging to work with.

Lyft [19] has released a dataset together with a devkit and fully annotated 3D bounding boxes. The dataset consists of approximately 80 GB of data comprising of images, video and LIDAR. In this study, we use the Lyft dataset for model training, application and validation.

# Methods

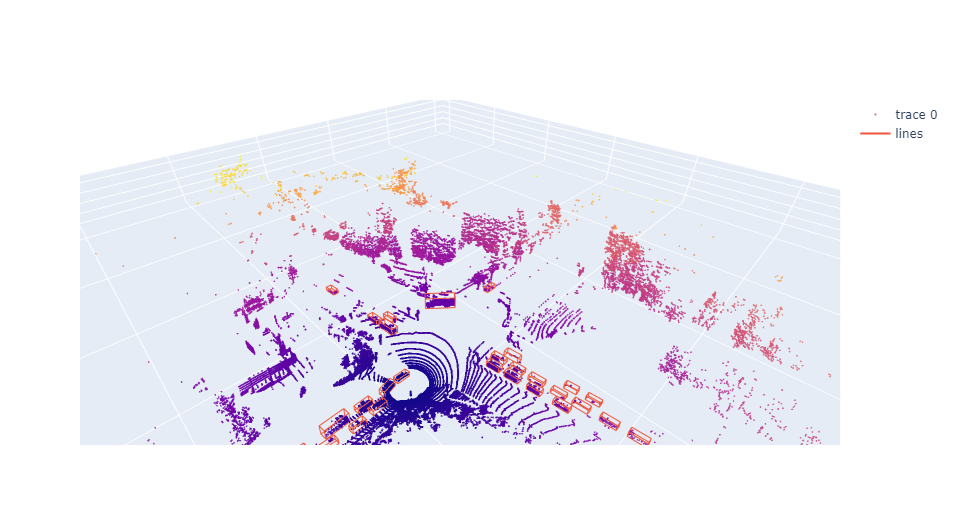
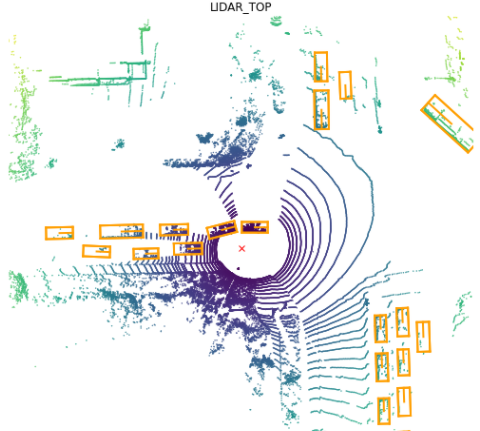
We build on employ different methods that are discussed as part of the literature review above. The approach in this paper is to apply the existing state of the art (rather than developing new state of the art methods) to the Lyft dataset and to learn about how the SDK functions with respect to autonomous vehicle sensor data collections. Another goal of the paper is to utilize and process LIDAR data for data visualization, model development for object detection and 3D bounding box projection.

## Dataset

As discussed above the dataset consists of 180 scenes (a collection of frames - essentially a short video) of about 25s each across different days in starting in January 2019 through May 2019, all located in Palo Alto, California. Each scene consists of approximately 126 samples. While scenes vary in the number of annotations (3D box and vehicle classification) they contain anywhere ranging from 350 annotations to over 5000 annotations on different scenes with average of 3,500 annotations per scene. Each annotation contains information on the category of the object and as well as a 3D box around the object indicating the volume occupied the object in 3D.

The figure below illustrates both LIDAR birds’ eye-view (BEV) and camera data for a scene, together with a 3D LIDAR visual representation of the data. LIDAR is represented as dots or points in 3D space and is therefore referred to as “point cloud”. In addition note the volumetric boxes around the objects that provide spatial information critical for autonomous vehicles navigating on the roadway. This analysis is focused on the LIDAR BEV representation although other representation and views are quite instructive as well.

Figure 8: Front Camera, LIDAR BEV and 3D Representation of a Sample

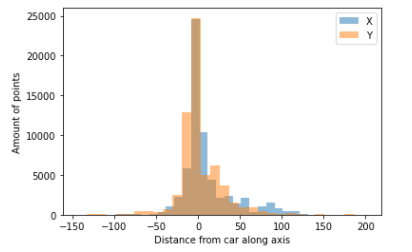
## Analysis

### Data Input

We experiment with two data sources. The first is the BEV data for each scene. As discussed previously, each scene results in approximately 126 BEV images. In the interest of time and training resources we reduce the training set to only 9 scenes. Due to memory constraints we limit the training the set further to 500 images.

The BEV view is based on LIDAR input as discussed previously. However is LIDAR is a point data and additionally as discussed the density of points is varies greatly, we voxelize the point cloud, that is to say we convert the point cloud into smaller three dimensional pixels that are then used to model and create BEV dataset rather than the actual number of LIDAR points. Additionally, the LIDAR points on a single LIDAR sweep is very high. For example on a single image we can have as many as 61,000 points (typically these can be as high as 100k). The histogram below shows X,Y coordinates (before voxelization)

Figure 9: Histogram of X,Y coordinates representing distance from the Car (ego-vehicle)



It maybe obvious therefore that the definition of the voxels (in terms of size in 3D space) is an important hyper-parameter of the model. In the analysis below therefore tweak the voxelization parameter and report two different set of model results.

### Network Architecture

We primarily investigate a U-Net architecture for model performance [9]. The U-Net architecture is described above as part of the literature review. We develop a simpler architecture that is less deep contains 4 layers of concatenation and only half the number of original filters as indicated in the paper (32 filters compared to 64 filters as indicated in the paper).

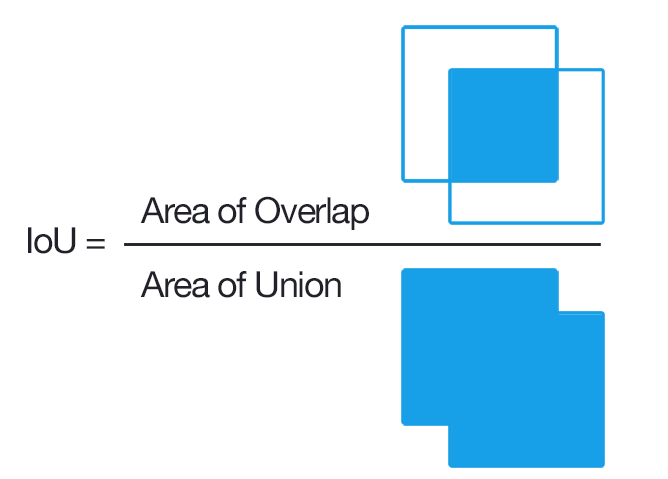
### Model Training

Model training is conducted using a batch size of 5 and trained for 5 epochs. On personal laptop with intel i-7 process with 4 CPU’s and 8GB of RAM, a single epoch takes upwards of 45 minutes to train. Training for 5 epochs takes approximately 4 hours to train. Using cloud services such as AWS or Google Cloud could significantly improve training times and processes. This approach was not investigated in this analysis but suggested as a future development exercise.

### Evaluation Metrics

A number of metrics are available that allow for the evaluation of the model. Two of the main metrics used are 1) Intersection over Union (IoU) and 2) the mean average precision. Intersection over union can be seen in the figure below, it is the area of intersection for the prediction bounding box and ground-truth bounding box divided by area of the union of the bounding box. IoU ranges from 0 to 1 with a typical threshold of 0.5.

Figure 10: Definition of Intersection over Union



Source: 22

Mean average precision is the average precision calculated over all classes [23]. The average precision is the area under the precision-recall curve. For reference precision and recall are defined as follows:

Precision=

Recall=

The true positive (TP) is based on different thresholds of IoU. Typically a threshold of 0.5 is used but different thresholds can be used.

## Result

We present the results of the analysis below. For the first dataset the model performance is not very good and can be improved significantly. Two factors play into this including the training set and the number of epochs used to train the model. The difference between the two datasets consists of the voxelization parameters that is mentioned as a hyper-parameter of the model. In dataset 1 we use coarser voxelization parameters, while in dataset 2, we use finer voxelization parameters (a factor of 2).

Table 1 shows the mAP overall and by class for the coarser voxelization. Voxelization used for this set consisted of x=0.4,y=0.4,z=1.5. For an IoU threshold of 0.25, mAP is 2% overall and for cars it is 18.6%. For the typical IoU threshold of 0.5, mAP reduces to 0.3%, with the car precision at 2.4%. At higher thresholds the mAP of the model reduces significantly.

Table 1: Mean Average Precision for Dataset 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | IoU Thresholds | | | |
| Metric / Class | 0.25 | 0.5 | 0.6 | 0.75 |
| mAP | 2.44x10-2 | 3.01x10-3 | 3.98x10-4 | 1.13x10-6 |
| Animal | 0.0 | 0.0 | 0.0 | 0.0 |
| Bicycle | 5.1x10-7 | 5.1x10-7 | 0.0 | 0.0 |
| Bus | 3.3x10-3 | 0.0 | 0.0 | 0.0 |
| Car | 1.86x10-1 | 2.4x10-2 | 3.18x10-3 | 9.11x10-6 |
| Motorcycle | 0.0 | 0.0 | 0.0 | 0.0 |
| Other Vehicle | 0.0 | 0.0 | 0.0 | 0.0 |
| Pedestrian | 4.7x10-5 | 5.69x10-6 | 1.97x10-6 | 0.0 |
| Truck | 0.6x10-3 | 0.0 | 0.0 | 0.0 |

Table 2 shows the mean average precision for the finer voxelization. The voxelization is finer / smaller by factor 2 in the X and Y direction (specifically the voxelization used X=0.2,Y=0.2, Z=2). A coarser voxelization is used in the Z direction. With finer voxelization the results show that the model performs better. At IoU of 0.25, mean precision for cars is 26% (compared to 18.6 with the coarser voxelization). However, for Trucks, the precision is lower, as the in the Z direction we use a coarser voxelization. At IoU of 0.5, however, the difference is greater. Mean precision across all classes is an order of magnitude better, and for cars it is approximately 4 times better.

Table 2: Mean Average Precision for Dataset 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | IoU Thresholds | | | |
| Metric / Class | 0.25 | 0.5 | 0.6 | 0.75 |
| mAP | 5.4x10-2 | 1.1x10-2 | 1.0x10-3 | 4.19x10-7 |
| Animal | 0.0 | 0.0 | 0.0 | 0.0 |
| Bicycle | 8.37x10-6 | 0.0 | 0.0 | 0.0 |
| Bus | 2.2x10-2 | 1.02x10-5 | 0.0 | 0.0 |
| Car | 2.6x10-1 | 7.89x10-2 | 7.05x10-3 | 2.93x10-6 |
| Motorcycle | 0.0 | 0.0 | 0.0 | 0.0 |
| Other Vehicle | 9.4x10-2 | 8.72x10-5 | 0.0 | 0.0 |
| Pedestrian | 8.14x10-6 | 0.0 | 0.0 | 0.0 |
| Truck | 9.2x10-4 | 0.0 | 0.0 | 0.0 |

## Discussion

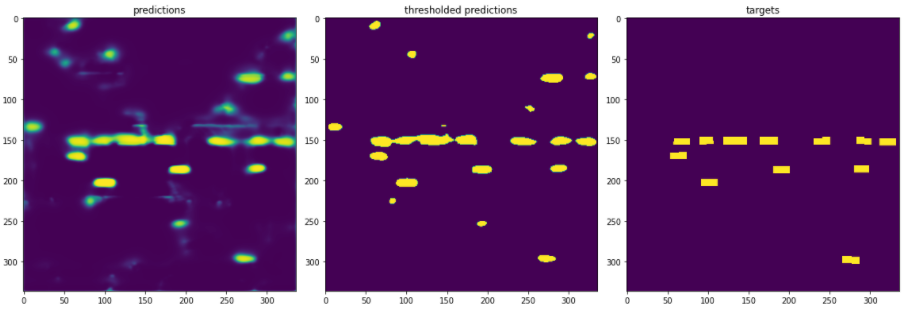
As shown form the analysis above and the literature review that there are many challenges associated with semantic segmentation using LIDAR data. Making the voxelization parameters finer can lead to a significant improvement in the model precision. A number of improvements can be potential to the existing. Some of these are listed below.

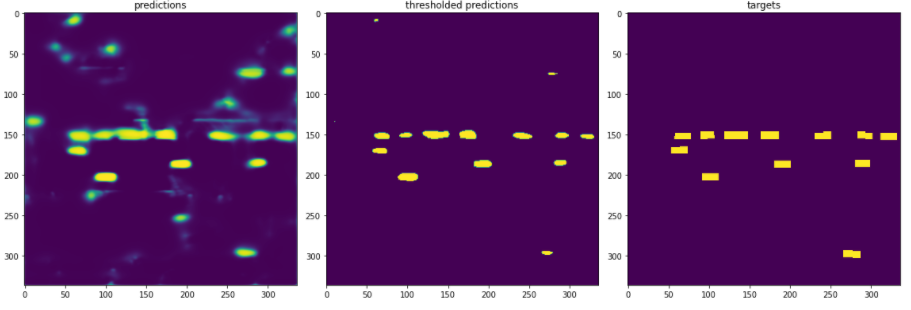
There a number of areas of improvement and future research that can be easily implemented. These include some of the initiatives and limitations already discussed namely:

1. Infrastructure Limitations: Given the large size of the datasets involved for this particular initiative utilizing cloud services would be advisable. Code from Jupyter notebooks from can then be run directly on the service using docker containers. Infrastructure limitations have greatly limited the size of the training dataset, batch size of the training set and the number of epochs that can be used for the training set. While the number of epochs used for training is the least impacted, the training time on a personal laptop can be significant.
2. Windows Environment & Parallelization: The Jupyter Notebook has a number of challenges when working in a windows environment. Ideal solution would be to either run in a linux based environment or cloud-based solution or running the code in native python environment.
3. Note the results from above. Truck and bus accuracy is low; probably due to height. Additional challenge is the wake due to the LIDAR point cloud. The wake of the point cloud causes vehicles that are close together to be classified as a single (much larger) vehicle even though there maybe two or three vehicle one after another.
4. 2D to 3D mapping: Bounding box estimates and the associated evaluation metrics are calculated in 3D while model training is done primarily in 2D using 2D representation via BEV imaging. In calculating bounding box projections in 3D from the 2D results obtained, we assume that all objects have the same height as the ego-vehicle. This assumption is probably why model fit on trucks and buses is so low (in addition to the fact that that the number of trucks and buses are much lower than cars). Height categorization needs to take into the account the different category of class that vehicles are estimated in 2D.
5. Filtering the Point Cloud: One challenge with the current methodology as implemented is that the LIDAR point cloud includes objects that are not vehicles, pedestrians or relevant roadway objects. While it would be difficult remove all non-relevant one way to remove less important data would be to remove all point cloud data not on currently on the current roadway based on the ego-vehicles position. This analysis could be developed by conducting morphological operations on the LIDAR point-cloud.

The figure below shows the impact and importance of filtering background form the analysis. The top figure shows a much a higher threshold for filtering background noise, while the bottom figure shows much lower threshold for filtering background noise. The middle figure (“thresholded predictions”) shows the difference between two (the other two columns, “predictions” and “targets”) are the same. Finally, we note that the background threshold used for the figure and in the results is based on an arbitrary threshold. These could be selected in a more structured way to minimize the associated error of selecting the threshold.

Figure 11: Background Threshold for the same BEV image (top=170,bottom=231)





# Conclusions

The use of LIDAR can be very useful in bounding box projection for 3D object recognition. However, there a number of challenges associated with developing good models to process and analyzing the data. Firstly, the datasets can be very large and thus require higher computing power. Secondly, memory requirements are significantly larger due to the spatial and pictorial nature of the datasets.

While this paper does not advance the state of the art, it does provide a good overview of the subject and suggests potential areas for further developing the analysis and areas for improvement.

## Potential Future Research

Areas for potential research include the following:

1. Experimenting with different training architectures. In this paper we primarily utilize a U-NET architecture however, a number of different architectures could be utilized including ResNET, FCN and utilization of Regional Proposal Networks (or RPN’s). These are all used in the literature for semantic segmentation tasks.
2. Using Pretrained models: We train models from the start rather than adopted / modifying weights of pre-trained models. Pretrained model weights are available in python packages such PyTorch and can be implemented for this analysis. Implementing should be relatively straight forward, however, it was not done here in the interest of available time.
3. Use of Sparse Tensors: The literature indicates that sparse tensors can be used to improve model processing times especially since most of the LIDAR point cloud is empty [16]. These should be explored further here to improve memory management and model inference times.
4. Adversarial Examples: Adversarial examples should be used to make the model results more robust.

# References

1. Li F, Krishna R., Xu D., Convolutional Neural Networks for Visual Recognition, Stanford 2020 (<https://cs231n.github.io/convolutional-networks/>)
2. Dumoulin V., Visin F., A Guide to Convolutional Arithmetic for Deep Learning, arXiv 2016 arXiv:1603.072285v1
3. Zeiler M., Krishnan D., Taylor G. and Fergus R. Deconvolutional Networks, CVPR 2010
4. Long J., Shelhamer E., Darrell T., Fully Convolutional Networks for Semantic Segmentation, arXiv 2015, arXiv: 1411.4038v2
5. Theano Development Team, 2008–2013, Fully Convolutional Networks for 2D Segmentation (<http://deeplearning.net/tutorial/fcn_2D_segm.html>)
6. Zeiler M., Taylor G., and Fergus R. Adaptive Deconvolutional Networks for Mid and High Level Feature Learning, International Conference on Computer Vision (ICCV) 2011
7. Goodfellow I., Shlens J., Explaining and Harnessing Adversarial Examples, International Conference on Learning Representation (ICLR) 2015
8. 2008–2013, Theano Development Team, Convolutional Neural Networks (LeNet) (<http://deeplearning.net/tutorial/lenet.html>)
9. Ronneburger O., Fisher P., Brox T., U-Net: Convolutional Networks for Biomedical Image Segmentation, arXiv 2015, arXiv 1505.04597v1
10. Graham B., Spatially Sparse Convolutional Neural Networks, arXiv 2014, arXiv: 1409.6070v1
11. Zhou Y. and Tuzel O., VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection, arXiv 2017, arXiv 1711.06396v1
12. Chen X., Ma H., Wan J., Li B., Xia T., Multi-view 3D Object Detection Network for Autonomous Driving, arXiv 2017, arXiv 1611.07759v3
13. Wang D., Posner I., Voting for Voting in Online Point Cloud Object Detection Proceedings of the Robotics: Science and Systems, Rome, Italy, 1317, 2015
14. Li B., Zhang T., Xia T., Vehicle Detection from 3D Lidar Using Fully Convolutional Network, arXiv 2016, arXiv:1608.07916v1
15. Zhu B., Jiang Z., Zhou X., Li Z., Gang Y., Class Balanced Grouping and Sampling for Point Cloud 3D Object Detection, arXiv 2019, arXiv: 1908.09492v1
16. Yan Y., Mao Y., Li B., SECOND: Sparsely Embedded Convolutional Detection, Sensors 2018
17. Ren S, He K., Girshcik R., and Sun J., Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks arXiv 2016, arXiv: 1506.01497v3
18. Geiger A., Lenz P., Urtasun R., Are we ready for Autonomous Driving? The Kitti Vision Benchmark Suite, Computer Vision and Pattern Recognition (CVPR) 2012
19. Kesten R. ,Usman M.,Houston J., Pandya, T., Nadhamuni K., Ferreira A., Yuan M., Low B., Jain A., Ondruska P., Omari S. , Shah S., Kulkarni A.,Kazakova A.,Tao C., Platinsky L., Jiang W. and Shet, V., Lyft 2019, Lyft Level 5 Perception Dataset 2020, <https://level5.lyft.com/dataset/>
20. Cordts M., Omran M., Ramos S., Rehfeld T., Enzweiler M., Benenson R., Franke U., Roth S. and Schiele B., The Cityscapes Dataset for Semantic Urban Scene Understanding 2016.
21. Caesar H., Bankiti V., Lang A., Vora S., Liong V., Xu Q., Krishnan A., Pan Y., Giancarlo B., Beijbom O., nuScenes: A multimodal dataset for autonomous driving, arXiv 2020, arXiv 1903.11027v5
22. Rosebrock A. Intersection over Union for Object Detection, 2016 (at <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/> accessed December 13, 2020)
23. Tan R. J. Breaking down Mean Average Precision (mAP), 2019 (at <https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52#1a59> accessed December 13, 2020).

Appendix:

Jupyter Notebook

<https://github.com/talham/research_capstone>