Fusing LiDAR and Camera Data for Advanced Context Recognition in Autonomous Navigation Sensory Systems Through Multidimensional Deep Neural Network Architectures

Emily S. Zhang

Mentored by Timothy J. Donahue, Cherry Creek High School

February 2021

Abstract

. Current robotic and vehicular systems for navigation, such as SLAM, rely heavily on sensory input to reconstruct and navigate environments but are unable to detect the structural integrity of objects. For example, Boston Dynamics' BigDog and Spot are equipped with both visual cameras and Light Detection and Ranging (LiDAR) sensors, but they have largely separate uses, and each cannot perform context recognition fast enough alone. LiDAR data cannot be used to accurately classify specific objects, while visual data cannot be used to find the position of individual objects other than by computationally expensive region generation methods, which fail regardless in low-light situations. Thus, this project proposes a method to fuse both LiDAR and camera data by using two different network architectures. First, LiDAR point cloud data is fed through a VoxelNet architecture to locate 3D bounding boxes around objects — this effectively performs object detection. Then, the 3D bounds are projected onto the 2D image space, and these regions of interest are ran through a separate convolutional neural network (CNN) based on the ResNet-101 architecture to classify the objects. Both networks were trained on the KITTI dataset, a widely-used autonomous driving dataset of both point cloud and camera information, so the CNN classifies objects into cars, pedestrians, cyclists, trucks, and vans. The VoxelNet reached an average precision (AP) of 75.52, 54.73, and 53.92 respectively for the easy, moderate, and hard occlusion levels of the KITTI dataset. The CNN attained an accuracy of 93.60% and an AP of 93.62, which outperforms previous metrics.

Contents

Introduction

1.1 Background

As technology improved through the 2010s, more and more efforts were made to build systems that could replicate human tasks, the most well-known involving autonomous driving and robotics. To effectively perform such a challenge, computer systems must have critical information about the environments around them, which they obtain using various sensors, such as visual (camera) systems, sound sensors, pressure sensors, etc. [1] Many methods have also been proposed to parse this data into useful observations for the system, the most popular for visual data being convolutional neural networks (CNN), which are deep neural networks designed for image analysis and classification [2].

One of the most prevalent high-level sensors, especially in navigational systems, is the Light Detection and Ranging, or LiDAR, scanner. LiDAR is a remote sensing method that uses light from a pulsed laser to measure variable distances to Earth and generates precise, three-dimensional information about different environments, often condensed and shown as a 3D point cloud [3]. Specifically, LiDAR measures the amount of time it take for emitted light to travel to the ground and back (this is further elaborated in the next section), and thus it remains largely accurate under varying lighting conditions, one of its main benefits to autonomous driving and robotics systems [4, 5]. However, LiDAR only provides a depth point cloud and does not contain sophisticated information about the structural integrity of specific objects.

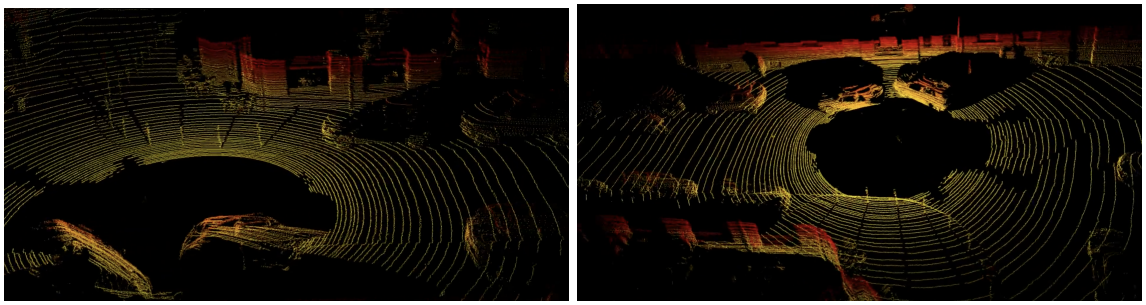

Figure 1.1: Visualization of KITTI LiDAR point cloud data through Mayavi

Naturally, the most popular form of data is visual, or camera, imaging. Most autonomous navigation systems contain some sort of camera. As opposed to LiDAR, camera data is able to provide high-resolution images for precise classification of singular objects or frames by existing CNN frameworks. However, due to the lack of 3D information, most camera-based methods of object detection first use a computationally expensive object proposal generation method, such as sliding-window. In addition, these methods generally suffer from varying illumination conditions and cannot accurately assess the correct orientation and geometry of 3D bounding boxes, even when they can accurately assess 2D boxes [6].

While many systems have both visual and LiDAR sensors, they often have largely separate uses. For example, Boston Dynamics's autonomous rough-terrain robot BigDog only utilizes its LiDAR functionality in a specific mode where the robot is following a human leader. In all other situations, it uses the vision system [7]. Recently, more work has been done in sensor fusion that attempts to reap the benefits of multiple forms of data. This work is elaborated on in the next section. This paper proposes another method of sensor fusion that uses two separate multi-dimensional networks to perform object detection and classification with LiDAR and camera data, respectively, in order to increase the situational awareness of the system.

1.2 KITTI 3D Object Evaluation Dataset

This paper utilizes the KITTI 3D Object Detection Evaluation Dataset to train and validate both neural networks. The KITTI Dataset is a highly reputable and cited autonomous vehicle dataset consisting of 7491 training images, 7518 test images, and their corresponding point clouds and calibration matrices. This comprises a total of 80,256 labeled objects of classes 'Car', 'Van', 'Truck', 'Pedestrian', 'Person (sitting)', 'Cyclist', 'Tram', and 'Misc' . The scenarios are diverse street scenes that reflect real-world traffic situations and areas [8]. The point clouds are obtained by a Velodyne HDL-64E scanner spinning at 10 frames per second, and the camera images are of size 1382 x 512 pixels. The recording platform is a Volkswagen Passat B6 [9].

In addition, depending on the size of the 2D bounding box in the image space and the occlusion levels, the data in the KITTI dataset are divided into difficulty levels of easy, moderate, and hard. Note that these are the levels referred to in the final results analysis in Section 3.1.

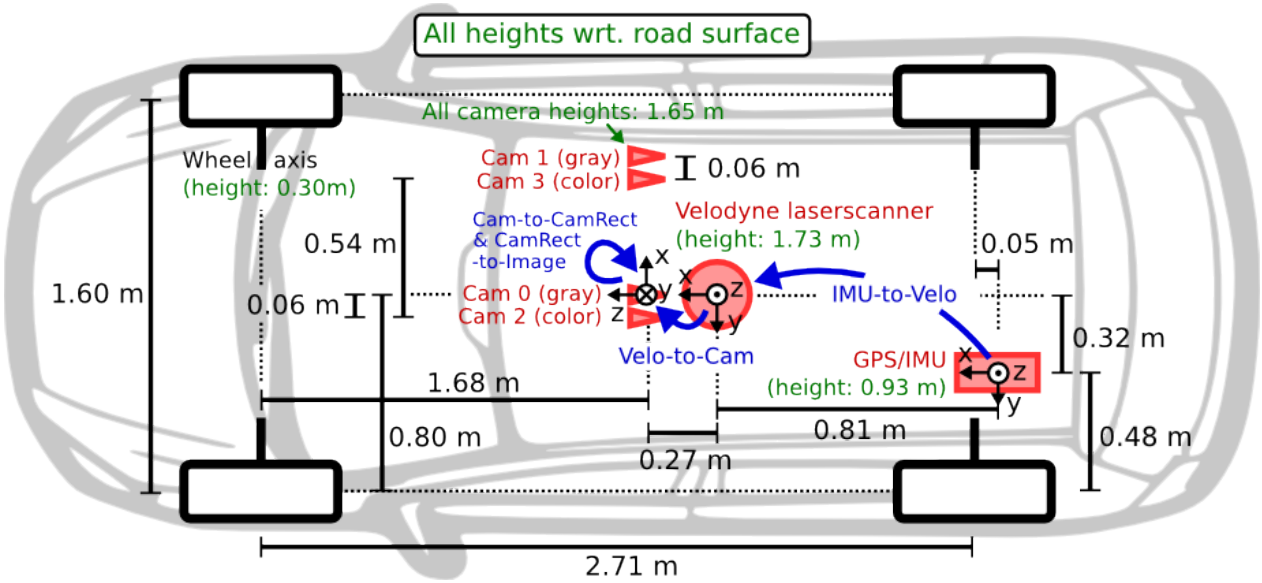

Figure 1.2: KITTI sensor setup top view [8]

1.3 VoxelNet

As briefly explained in section 1.1, there has been considerable work done in different methods of evaluating visual data, such as CNNs and sliding-window methods. However, there has been much less previous work done in evaluating 3D data like LiDAR point cloud data. Most developments in this field have been relatively

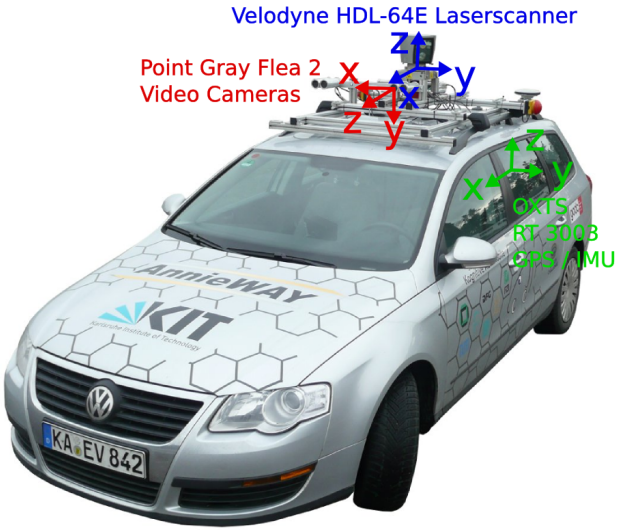

Figure 1.3: KITTI fully equipped vehicle [8]

recent. This project aims to find 3D bounding boxes from LiDAR alone (which are later projected onto the 2D image space), for which a recently developed architecture, VoxelNet[10], is used.

The VoxelNet architecture is constituted of mainly three blocks: the feature learning network, convolutional middle layers, and the region proposal network. Figure 1.4 is a diagram provided by the authors showing the overall VoxelNet structure.

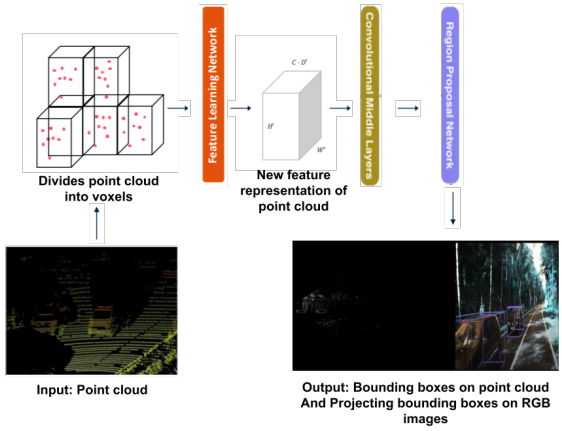

Figure 1.4: VoxelNet architecture [10]

The feature learning network extracts features from the point cloud data. To do this, the point cloud is first partitioned into equally spaced voxels. The points are grouped according to the voxel they are assigned to. Furthermore, voxels with less than a threshold of LiDAR points are automatically omitted from processing, which decreases computational load and imbalance. With each remaining voxel, the local mean is computed as the centroid of all the points within the voxel grid. Each point $\boldsymbol{p_i} = [x_i, y_i, z_i, r_i]$ (where r is the reflectance value) in the voxel is then augmented with the offset of the local mean. In particular, this is the set of points $\{\hat{\boldsymbol{p}}_{\boldsymbol{i}} = [x_i, y_i, z_i, r_i, x_i - v_x, y_i - v_y, z_i - v_z]\}$ for each voxel, where the centroid of all the points p_i in the particular voxel is denoted (v_x, v_y, v_z). After transformation, each $\hat{\boldsymbol{p}}_{\boldsymbol{i}}$ is further fed to a fully connected neural network, which has a Linear layer, batch normalization, and rectified linear unit

(ReLU) layer. This transforms each $\hat{p}_i$ into a feature space. Next, element-wise max-pooling is used to obtain the locally-aggregated features for each voxel from the point-wise features, which encodes the shape of the surface within the voxel. Lastly, point-wise concatenation is used to aggregate point-wise features with locally-aggregated features. Figure 1.5 is a diagram by the authors that show the voxel feature encoding layer of the feature learning network.

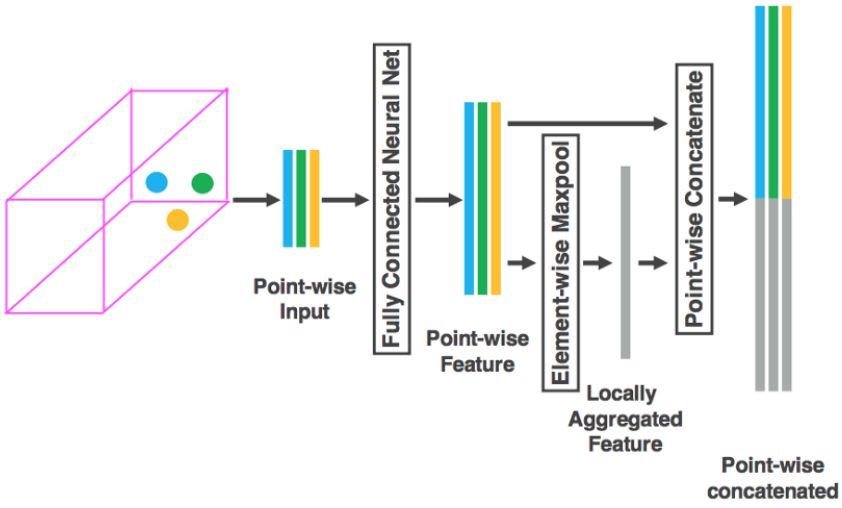

Figure 1.5: Feature learning network on a singular voxel [10]

The convolutional middle layers convert the voxel features previously obtained to dense 4D feature maps at a reduced size using convolution, batch normalization, and ReLU Layers.

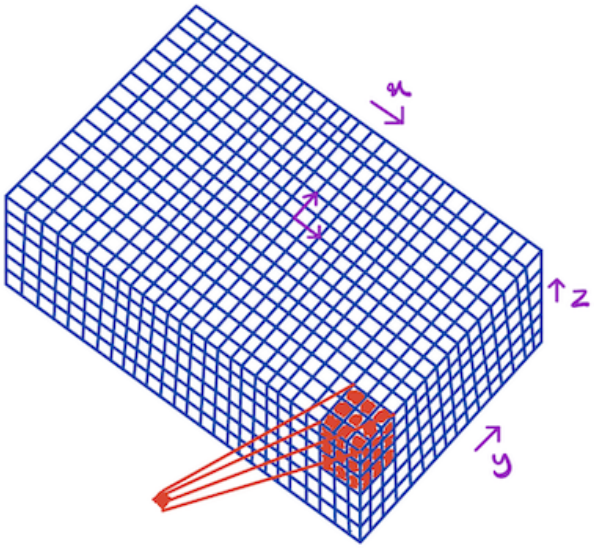

Figure 1.6: Visualization of 3D voxel grid convolution [11]

The region proposal network is the final portion of the VoxelNet architecture. It contains three blocks of fully convolutional layers. Each follows a similar structure at decreasing sizes: the first layer of each downsamples the feature map by half with a stride size 2, and is followed by a series of convolutions with a stride of 1. After each of these convolutional layers, batch normalization and ReLU are applied. The output

of each block is then upsampled to a fixed size (detailed in Figure 1.7) and concatenated to construct the high-resolution feature map. Lastly, the feature map is mapped to a probability score map and a regression map, which effectively output bounding box predictions.

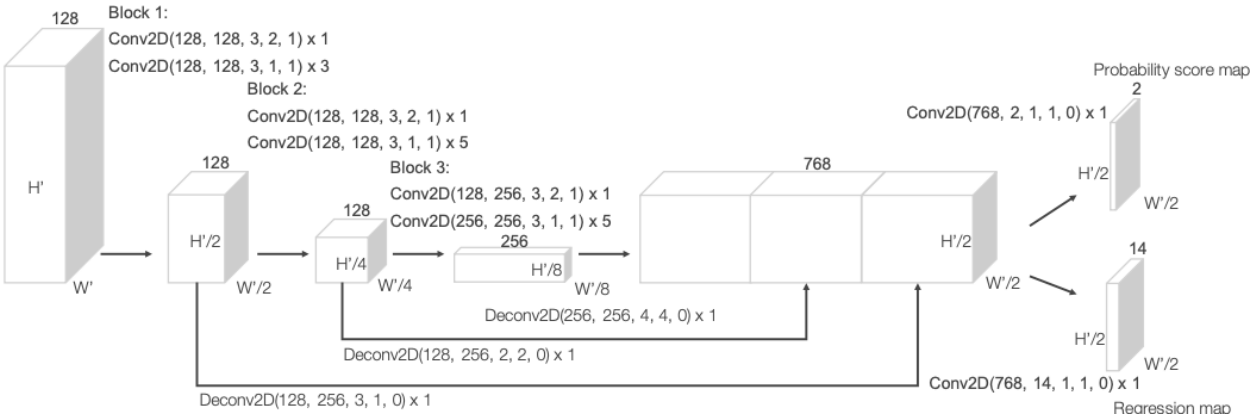

Figure 1.7: VoxelNet region proposal network architecture [10]

The loss function for training of VoxelNet is also quite unique and requires some explanation. Note that this is the loss function visualized in Section 3.1. Let the set of N_{pos} positive anchors and N_{neg} negative anchors be denotes as $\{a_i^{pos}\}$ and $\{a_j^{neg}\}$, respectively. Let the ground truth box be $\{x_c^g, y_c^g, z_c^g, l^g, w^g, h^g, \theta^g\}$, where x_c^g, y_c^g, z_c^g represent the center location, l^g, w^g, h^g are the dimensions of the box, and θ^g is the yaw rotation around the Z-axis. For each positive anchor $(x_c^a, y_c^a, z_c^a, l^a, w^a, h^a, \theta^a)$, the residual vector $\boldsymbol{u^*} = [\Delta x, \Delta y, \Delta z, \Delta l, \Delta w, \Delta h, \Delta\theta]$ is defined, where:

$$\Delta x = \frac{x_c^g - x_c^a}{d^a} \tag{1.1}$$

$$\Delta y = \frac{y_c^g - y_c^a}{d^a} \tag{1.2}$$

$$\Delta z = \frac{z_c^g - z_c^a}{d^a} \tag{1.3}$$

$$\Delta l = \log(\frac{l^g}{l^a}) \tag{1.4}$$

$$\Delta w = \log(\frac{w^g}{w^a}) \tag{1.5}$$

$$\Delta h = \log(\frac{h^g}{h^a}) \tag{1.6}$$

$$\Delta\theta = \theta^g - \theta^a \tag{1.7}$$

where $d^a = \sqrt{(l^a)^2 + (w^a)^2}$, or the diagonal of the anchor box base. Letting p_i^{pos} and p_j^{neg} be the softmax output for a_i^{pos} and a_j^{neg} respectively, $\boldsymbol{u_i}$ be the regression output, L_{cls} be the binary cross entropy loss, and L_{reg} be the regression loss obtained by the smooth-L1 function [12, 13], the loss function is:

$$L = 1.5 \cdot \frac{1}{N_{pos}} \sum_i L_{cls}(p_i^{pos}, 1) + \frac{1}{N_{neg}} \sum_j L_{cls}(p_j^{neg}, 0) + \frac{1}{N_{pos}} \sum_i L_{reg}(\boldsymbol{u_i}, \boldsymbol{u_i^*}) \tag{1.8}$$

The coefficients of 1.5 and 1 of the first two terms balance the relative importance between the positive and negative classification loss. For clarification, the smooth-L1 loss function is originally used for Fast R-CNN and is defined as follows:

$$L_{reg}(\boldsymbol{u_i}, \boldsymbol{u_i^*}) = \sum_j \text{smooth}_{L_1}(\boldsymbol{u_{i,j}^*} - \boldsymbol{u_{i,j}}) \tag{1.9}$$

where j runs through all elements of the vector and

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{1.10}$$

This is a full description of the VoxelNet loss equation used in the original paper.

1.4 Visual-CNN and ResNet

In recent years, visual convolutional neural networks (CNNs) have become increasingly advanced in the field of image classification. A convolutional neural network is a deep learning algorithm that can extract features from an input image and accurately identify the overall object from these features. This paper will begin by discussing the basic makeup of all visual-CNNs and then explain the specific features of ResNet-101, which is the network architecture trained in this project.

Given an RGB input image, the CNN's goal is to reduce this image into a form that is easier to process and less computationally-intensive, without losing the distinguishing features [14]. The first layer is a convolutional layer, which is the core of the neural net. A filter of a predetermined size by a certain stride is slid over the image, performing matrix multiplication between the filter and the portion of the image it is covering at the moment.

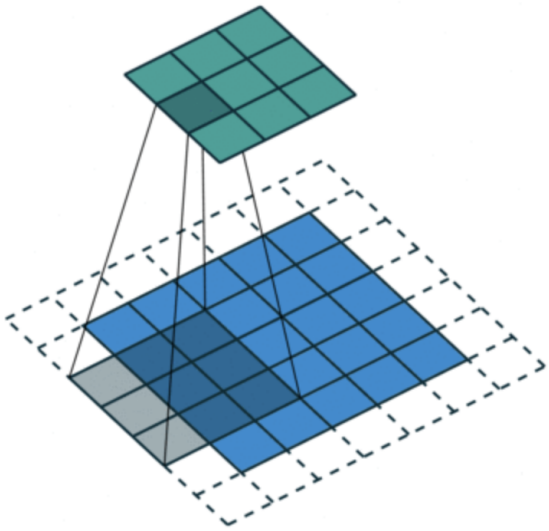

Figure 1.8: Convolution operation with padding on edges [14]

Most CNNs, and the ResNet-101, have many convolution layers. The earlier ones pick up low-level features, such as edges, and the later ones pick up high-level features, such as objects and orientation.

Next, there is a Rectified Non-Linearity Unit, or ReLU, layer, which is an activation function that introduces non-linearity into the model. It can be written as $f(x) = \max(0, x)$ and is applied on the result of the previous convolution [15].

The last stage of each block is the pooling layer, which reduces the spatial size of the convolved feature in order to decrease computational power and time. The most popular type of pooling is max pooling, which

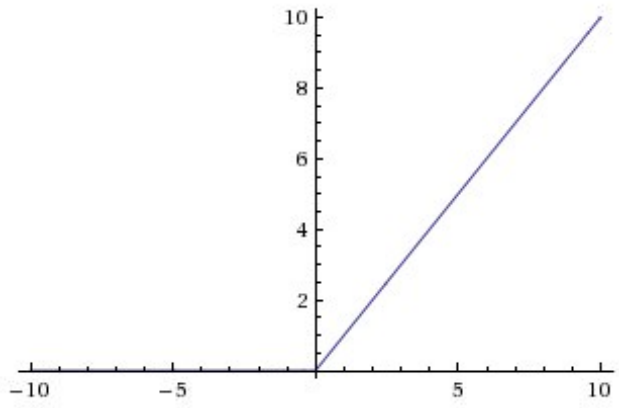

Figure 1.9: Graphical representation of ReLU

returns the maximum value from the portion of the image covered by the kernel. While there are other types of pooling, max pooling is most effective in retaining important features, and it is also the type of pooling used in ResNet.

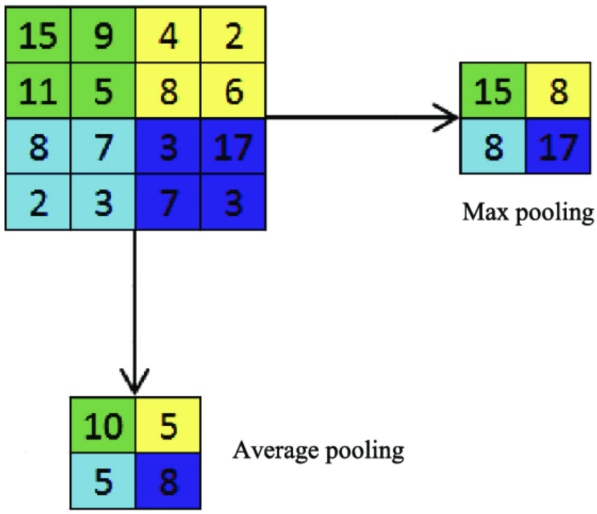

Figure 1.10: Example of max and average pooling [16]

Finally, the fully-connected (FC) layer is the last layer in the whole architecture, and it is used for classifying the complex features extracted from the previous layers. The image is flattened to be passed into the fully-connected layer, which is trained using back-propagation. The final output is calculated by softmax, which gives a vector of values between zero and one that represent the probabilities for each class [17].

For this project, the ResNet-101 architecture, the 101-layer version of the winner of ILSCRV 2015, is transfer-learned. The ResNet implements a skip connection to fit the input from the previous layer to the next layer without any modification (shown in Figure 1.12, which alleviates the problem of vanishing or exploding gradients during back-propagation [18].

The ResNet also uses a bottleneck design to reduce time complexity, especially for the ResNet-101 and 152 versions, as well as uses batch normalization after convolutional layers. It is originally trained on the ImageNet dataset, which has approximately 1.2 million training images, 50,000 validation images, and 1000 categories. On this dataset, ResNet outperforms all other state-of-the-art CNN architectures, as shown in

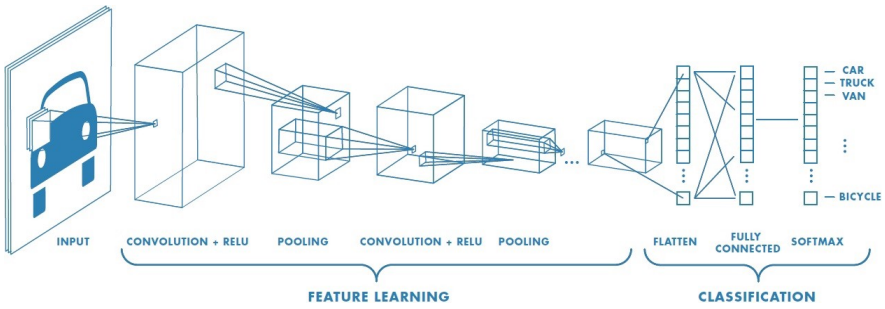

Figure 1.11: Basic CNN architecture [14]

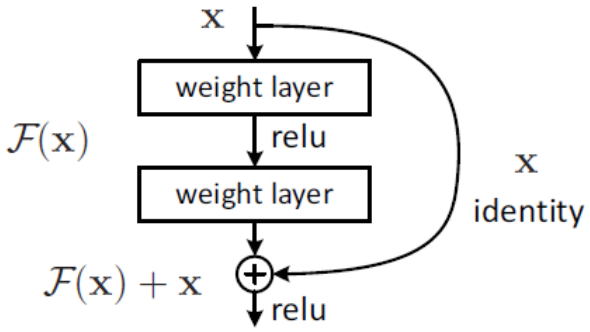

Figure 1.12: Building Block of ResNet, including skip connection [18]

Figure 1.13.

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Figure 1.13: Error rates of single model results on the ImageNet set [18]

Transfer learning technology is relatively new. It is an approach where a pre-trained model is used as the starting point for training on a new dataset. In this paper, a pre-trained ResNet-101 with weights from ImageNet is used. This is detailed more in Section 2.3.

1.5 Research Question

Can VoxelNet and a visual-CNN be combined to give robotic sensory systems advanced context knowledge from depth and camera data?

This solution would allow computer systems to perform accurate object detection and classification of surroundings, which is critical for navigational and driving applications. In addition, improved situational awareness is necessary for robotics in areas such as healthcare and natural disaster aftermath. The fusion aspect allows more versatility into the model, since the system will no longer be reliant on only one sensor – even in low-light situations, the LiDAR would still be able to perform object detection. Development of a uniform system as this project attempts is necessary for robotics to have autonomy. This project builds on previous research by exploring a potential autonomous driving application of 3D and 2D neural networks.

Methods and Materials

2.1 Dataset Processing

For this project, the first step was to process the data required to train the 3D VoxelNet architecture. The 3D point clouds were projected onto the image space and the points that existed outside of the image space were omitted from the Velodyne files. For training the VoxelNet, the data was split into training and validation sets as specified by KITTI.

Next, to process the data to train the CNN through transfer-learning, the object bounding boxes and the calibration matrices given in the dataset were used to crop all individual objects of the types 'Car', 'Truck', 'Pedestrian', 'Truck' and 'Cyclist' from the 2D image space (class diversity has been limited from the available eight classes in the KITTI dataset).

Figure 2.1: Examples of data used for training the visual CNN

Shown in Figure 2.2 are the counts of individual objects in each class from the KITTI dataset. If the ResNet-101 is trained directly on this data, the network would be highly skewed toward the 'Car' class. Thus, the categories are balanced by only using 1000 of each type.

2.2 VoxelNet

The VoxelNet was implemented in Python with the Keras API as detailed in Section 1.3 and the original paper. Voxel size follows the recommended parameters $v_D = 0.4$, $v_H = 0.2$, $v_W = 0.2$ meters, and voxels are eliminated from consideration when there are less than 45 point values, which has been found to be an optimal threshold for the KITTI dataset and Velodyne scanner in particular [10]. Through experimentation, it is found for this project that the 3D bounding box anchor size $l^a = 3.5$, $w^a = 1.5, h^a = 1.7$ generally includes cars, trucks, pedestrians, vans, and cyclists. This is centered at $z_c^a = -0.8$ with 0 and degree rotation. An

Classification	count
Car	28741
DontCare	11295
Pedestrian	4486
Van	2914
Cyclist	1627
Truck	1094
Misc	973
Tram	511
Person_sitting	222

Figure 2.2: Counts of each classification after initial 2D cropping of KITTI dataset

anchor is assigned as positive if its IoU (Intersection over Union) with ground truth is greater than 0.5. An anchor is assigned as negative if its IoU with ground truth is less than 0.35.

To reduce overfitting in training the model with less than 4000 point clouds, three type of data augmentation are used [10]. The first type applies perturbation to each independent ground truth 3D bounding box and the LiDAR points inside that box, rotating by a uniformally distributed $\Delta\theta \in [-\frac{\pi}{10}, +\frac{\pi}{10}]$ and translating by $(\Delta x, \Delta y, \Delta z)$, where Δx, Δy, Δz are chosen independently from a Gaussian distribution with a mean of 0 and a standard deviation of 1. Physically impossible outcomes are then reverted. The second and third types of augmentation are simply global scaling (by a uniform factor in [0.95, 1.05]) and global rotation (by a uniform angle in $[-\frac{\pi}{4}, +\frac{\pi}{4}]$). These augmentation effectively allow the model to learn from more variations in orientation, size, and distance.

The learning rate used for stochastic gradient descent was 0.01 for the first 30 epochs and 0.001 for the last 5 epochs. The importance of learning rate is explained in CNN section 2.3. The batch size used was 16 point clouds.

In order to use faster GPUs to train the 3D network, the network was trained in Google Colab Pro. Visualization was done in TensorFlow.

2.3 Convolutional Neural Network

For transfer learning, the Fast.ai database is used to simplify parts of the implementation. First, only the last layer group of the pre-trained ResNet-101 is trained on the processed KITTI pictures of cars, vans, pedestrians, cyclists, and trucks. Then, the all the previous pre-trained layers are unfrozen for the weights to be trained specifically for classifying our KITTI classes.

As for the learning rate of the CNN, which is critically important for training the network most effectively over the least number of epochs, the network is trained using a 1cycle policy, where learning rate is varied throughout the training over multiple epochs.

Figure 2.3: Effects of learning rates on CNN training [19]

To do this, the network is trained for a few epochs at a learning rate that starts small and is increased after each mini-batch until the loss value begins to explode. This effectively finds the maximum learning rate, which should be an order of 10 lower than the learning rate where the loss is minimum. The 1cycle policy recommends cycling the learning rate between lower and upper bound during the run. The upper bound is the maximum learning rate just found, and the minimum learning rate is one magnitude of 10 lower. In particular, cycles are two steps of equal lengths, one going from the minimum to the maximum, then another going from that maximum to the minimum again. The length of this cycle should be slightly less than the total number of epochs, so that in the last part of the training, the learning rate should be allowed to decrease more than the minimum [20].

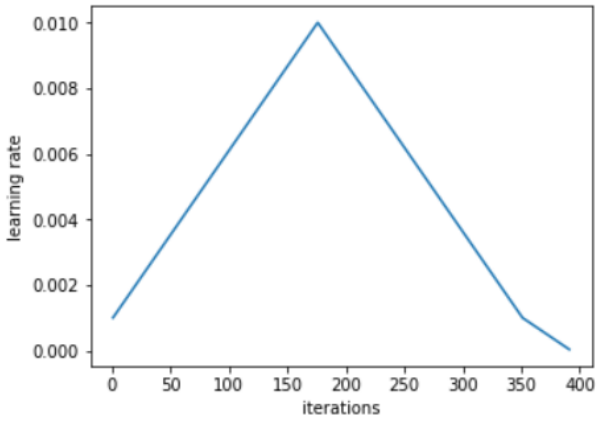

Figure 2.4: Example of 1cycle cyclic learning rate over the entire training run

The motivation and benefit behind this approach is that during the middle of the training, high learning rates will as a regularization method and prevent the model from overfitting. Specifically, it prevents the model from landing in a steep area of the loss function, preferring to find a flatter minimum. Then, during the last part of the training, descending learning rates until annihilation allows the network to go inside a steeper local minimum within the smoother area.

Along with the cyclic learning rate, the momentum, which helps accelerate gradient vectors in the right direction and thus leads to faster converging, is also cyclic under the 1cycle policy. However, when the learning rate increases, the momentum decreases, and vice versa, to lead to better results. This is since the

stochastic gradient descent should go quickly to find a flatter area, so the new gradients need to be given more weight [20]. In practice, a maximum momentum of .95 and minimum momentum of .85 are used.

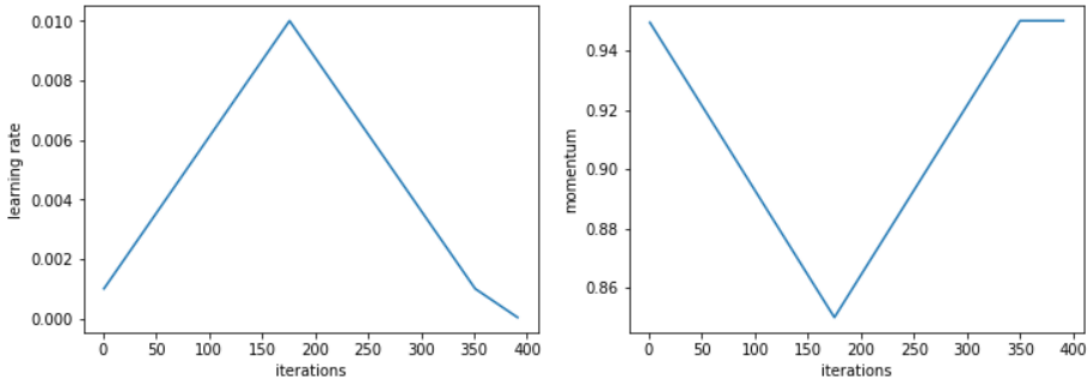

Figure 2.5: Example of 1cycle cyclic learning rate and momentum in relation to each other

Beyond learning rates, the dataset is first reduced to a uniform size of 460x460. Then, a series of flip, rotate, zoom, warp, and lighting augmentations are done on the data, and then the data is cropped to a 224x224 size. This is a method called presizing, which reduces computational load but still gives a desired result. Data is also normalized using the ImageNet statistics. The batch size used is 32. This is larger than the batch size of the VoxelNet, since the CNN is computationally less intensive to train.

Results

3.1 VoxelNet Results

After training the VoxelNet for 35 epochs, the training and validation loss graphs are shown in Figure 3.1.

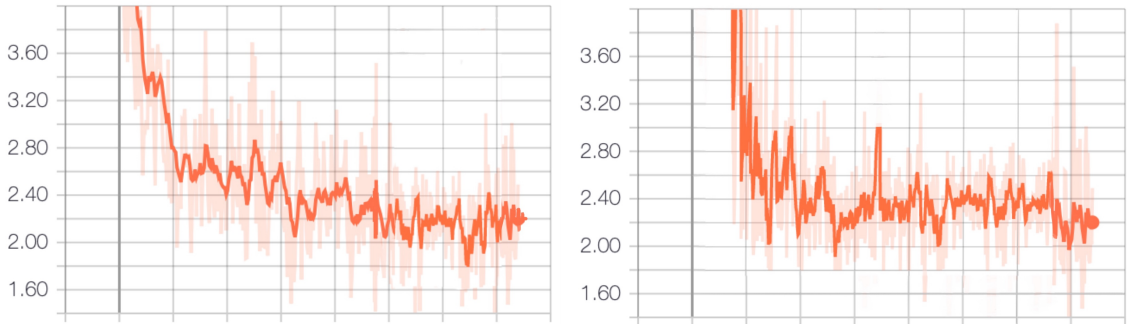

Figure 3.1: Training and validation loss graphs over 35 epochs of VoxelNet training

For metrics, note that the IoU threshold for 'Car', 'Van', and 'Truck' is 0.7, while the IoU threshold for 'Pedestrian' and 'Cyclist' is 0.4 (in bird's eye view). This is since the bounding boxes used are slightly larger than that of pedestrians and cyclists, and the efficacy of the system does not depend on the VoxelNet perfectly enclosing the object in the predicted 3D bounding box. Because of this specific use case, the metric evaluation deviates from the usual KITTI protocol.

Table 3.1 shows the average precision (AP) for the three types of KITTI occlusion levels after 55 epochs of training.

	Easy	**Moderate**	**Hard**
Average Precision	75.52	54.73	53.92

Table 3.1: AP of VoxelNet after training

The average inference time for the VoxelNet network was 253 ms.

3.2 CNN Results

As mentioned in Section 2.3, this project first finds the ideal learning rate for cyclic training. The graph generated from gradually increasing the learning rate at each mini-batch for an epoch until the loss explodes is shown in Figure 3.2.

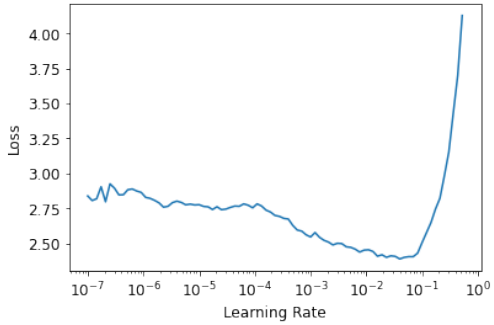

Figure 3.2: Graph of the loss vs. learning rate when tested for an epoch

Using this graph, the maximum learning rate for training the last layers of the pre-trained ResNet-101 is then determined to be $3 \cdot 10^{-3}$, which is before the loss explodes and while it is still decreasing.

Using this as the maximum learning rate, the network is then trained for 60 consecutive epochs. Note that currently, only the last layers of the CNN are unfrozen and only those weights are updated with the KITTI data. Shown in Figure 3.3 is the training and validation loss graph for this first stage of training.

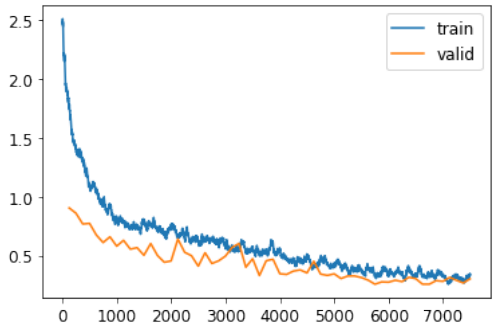

Figure 3.3: Training and validation loss graph for stage one of CNN training

The momentum is also cyclic (see Section 2.3). Figure 3.4 includes the graphs of the cyclic learning rate and momentum over the entire training process.

Select metrics of the ResNet-101 after the first stage of training are shown in Table 3.2.

Training Loss	**Validation Loss**	**Accuracy**	**Precision**	**Recall**	**F1-score**
.3259	.2580	.9170	.9197	.9174	.9167

Table 3.2: Metrics of the ResNet-101 after stage one of transfer learning

The 91.70% accuracy and .9167 F1-score already suggested the network had attained a high level of classification ability on the KITTI dataset. From here, the weights on all of the pre-trained layers, not just the last layers, are unfrozen to be updated on the dataset. New ideal minimum and maximum learning rates

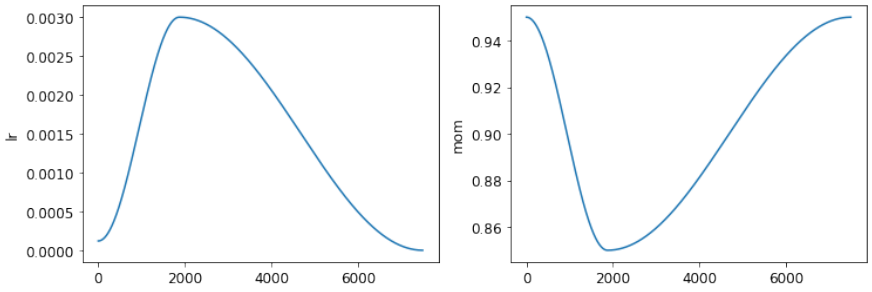

Figure 3.4: Learning rate and momentum graphs over stage one of training

now must be found on the newly trained network. The new graph from increasing the learning rate over one epoch is shown in Figure 3.5.

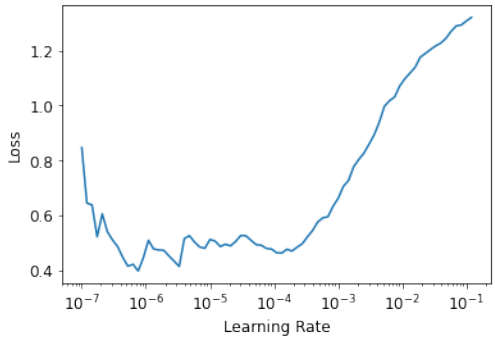

Figure 3.5: Graph of the loss vs. learning rate when tested for an epoch after stage one training

Thus, maximum learning rate is taken to be 10^{-5}, and the minimum learning rate is set to be 10^{-6}. The CNN is trained for 100 consecutive epochs. The training and validation loss over this stage two of training are shown in Figure 3.6. The cyclic learning rate and momentum are shown in Figure 3.7.

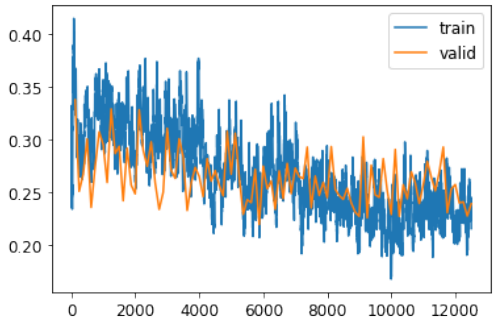

Figure 3.6: Training and validation loss graph for stage two of training the CNN

There is less drastic improvement in the losses compared to during stage one, which is expected because the last layers, which capture high-level features and details of the new dataset, have already been trained. However, stage two still helps the model improve. Table 3.3 shows the new metrics after stage two.

The accuracy increased to 93.60% from 91.70%, and the F1-score increased to .9357 from .9167. The average inference time for the network is 32 ms. The confusion matrix for the final trained ResNet-101 after both stages is shown in Figure 3.8.

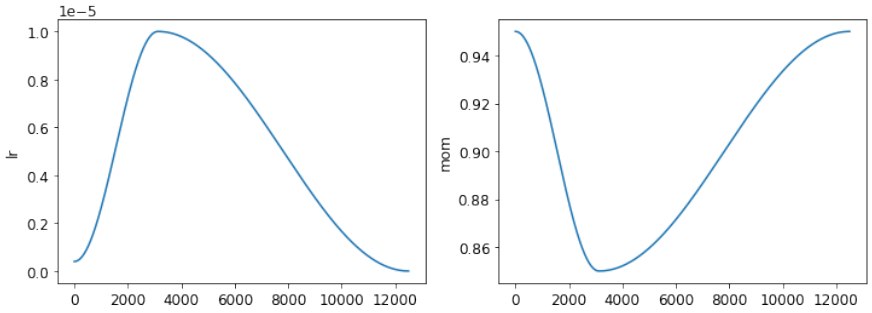

Figure 3.7: Cyclic learning rate and momentum graphs over stage two

Training Loss	**Validation Loss**	**Accuracy**	**Precision**	**Recall**	**F1-score**
.2238	.2262	.9360	.9362	.9364	.9357

Table 3.3: Metrics of the ResNet-101 after stage two of transfer learning

For the CNN, Table 3.4 compares accuracy by class metrics with a recent KITTI object classification algorithm with a combined region proposal and CNN system which classified between the classes 'Pedestrian' and 'Car' [6]. The CNN was a VGG-16 model pre-trained on the ImageNet dataset, which makes it optimal for comparison. It was not trained with cyclic learning rates and/or momentum.

	Car	**Pedestrian**	**Cyclist**	**Truck**	**Van**
My model	**.9505**	**.9560**	**.9223**	**.9760**	**.8514**
Comparison model	.8904	.7818			

Table 3.4: Comparison of accuracy by class between my model and an alternative algorithm

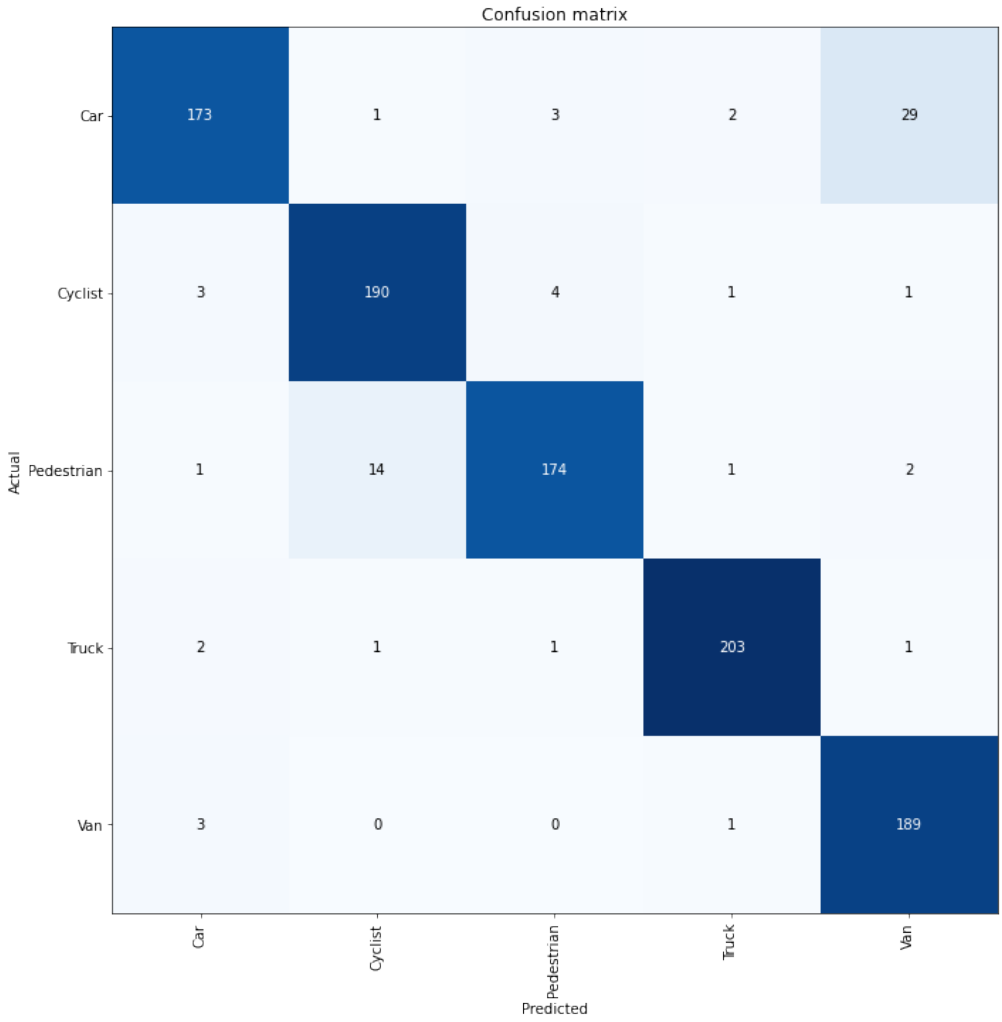

Figure 3.8: Confusion matrix of the ResNet-101 for the five classes

Conclusion and Future Work

Overall, the project was a success. The project criteria were met: the VoxelNet was successfully constructed and fully trained to output 3D bounding boxes, and the CNN was successfully transfer-learned with ideal cyclic learning rate and momentum. The VoxelNet was measured for average precision, differentiated by KITTI occlusion level, and inference time. The ResNet-101 CNN was measured for accuracy, precision, recall, F1-score, and inference time. Accuracy was further differentiated by the five classes 'Car', 'Truck', 'Pedestrian', 'Truck' and 'Cyclist'. The usage of cyclic learning rates not only outperformed previous accuracy of 'Car' and 'Pedestrian' classification, but also allowed the network to classify the three additional classes at high accuracy. The training and validation loss graphs were plotted for both networks and suggest steadily learning architectures without sign of much overfitting. Thus, it can be concluded that a multi-network method can be used to increase robotic sensor systems' advanced context knowledge.

There were a few limitations to the project and its end result. First, the VoxelNet was only trained for 35 epochs due to the lack of computational power and time, even with the Colab high-performance GPU, since each epoch took approximately 4 hours to complete. Ideally, the network would have been trained for 160 epochs. The inability to do this likely led to the lower AP scores, especially for the more occluded point clouds. While the visual-CNN had less of an issue with this, due to the use of cyclic learning rate and transfer learning, which substantially decreased the number of epochs needed for good classification accuracy, the model did still have room for improvement, as the training and validation loss graphs suggest. Thus, the results of both networks could be improved substantially if more time and power was used to train, such as in academic machine learning research, where multiple clusters of GPUs are used over months to fully train models. High school research evidently does not have the resources to achieve this.

While the research in general was a success, there are multiple facets for future work to be done, which include:

- Experimenting with different anchor box parameters and hyperparameters for the VoxelNet
- Experimenting with visual-CNN architectures other than ResNet-101
- Attempting to classify for any of the other three classes in the KITTI dataset
- Attempting to apply the multi-network method introduced in this paper to other autonomous sensory applications, such as working with household objects like chairs, tables, plants, etc. (this would require a different dataset or curation of a new dataset with LiDAR and visual information)
- Using more high-performance GPUs over a longer period of time to fully train the models to completion

References

[1] Different types of sensors and their uses. https://www.thomasnet.com/articles/instruments-controls/sensors/.

[2] Convolutional neural network. https://www.sciencedirect.com/topics/engineering/convolutional-neural-network.

[3] What is lidar and what is it used for? https://www.americangeosciences.org/critical-issues/faq/what-lidar-and-what-it-used, Nov 2020.

[4] Leah A. Wasser. The basics of lidar - light detection and ranging - remote sensing. https://www.neonscience.org/resources/learning-hub/tutorials/lidar-basics, Oct 2020.

[5] P. Sudhakar, K. A. Sheela, and M. Satyanarayana. Imaging lidar system for night vision and surveillance applications. In *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 1–6, 2017.

[6] X. Zhao, P. Sun, Z. Xu, H. Min, and H. Yu. Fusion of 3d lidar and camera data for object detection in autonomous vehicle applications. *IEEE Sensors Journal*, 20(9):4901–4913, 2020.

[7] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. Bigdog, the rough-terrain quadruped robot. *IFAC Proceedings Volumes*, 41(2):10822–10825, 2008. 17th IFAC World Congress.

[8] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[9] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[10] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017.

[11] Mohammad Sanatkar. Lidar 3d object detection methods, Jun 2020.

[12] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[13] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[14] Sumit Saha. A comprehensive guide to convolutional neural networks. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, Dec 2018.

[15] Dan S. Becker. Rectified linear units (relu) in deep learning. https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning, May 2018.

[16] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29:1–98, 06 2017.

[17] Softmax function. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer, May 2019.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[19] Jeremy Jordan. Setting the learning rate of your neural network. https://www.jeremyjordan.me/nn-learning-rate/, Aug 2020.

[20] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018.

* All uncited images, graphs, and tables are created by the student.