

Object Detection in Autonomous Driving Vehicles

Adil Sadik, Qianli Song, Rui Chen
Department of Computer Science
Stanford University

asadik@stanford.edu, cr1990@stanford.edu, qlsong@stanford.edu

Abstract

In this final project, we have worked on object detection and segmentation for autonomous driving vehicles. Here, we mainly use two types of dataset inputs. The first type is the group of snapshot images taken when the vehicle is driving down the road. The second type is the LIDAR data points collected by the LIDAR sensors mounted on the vehicle. The outputs are groups of predicted 3D bounding volumes, which stand for the location, type and size of objects in the images. Our goal is to achieve accurate predictions in these predicted 3D bounding volumes on objects. The final result is evaluated using mean average precision at different IoU (intersection over union) thresholds. In the Related Work section, we have reviewed object detection algorithms using CNNs, such as R-CNN and Mask R-CNN, and object detection algorithms using LIDAR point clouds, such as MV3D, PointNet and VoxelNet.

We have used three different algorithms for object detection. The first algorithm is YOLO, which stands for You Only Look Once. YOLO is a 2D object detection algorithm, and we have used pre-trained weights of YOLOv3 for object detection. Since we don't need to do any training here, we mainly used the YOLO algorithm for qualitative analysis. The second algorithm is the PointPillars network, which is an encoder that utilizes PointNets to learn a representation of point clouds organized in vertical columns (pillars). The overall architecture of Point Pillars contains three components: a pillar feature net, a backbone 2D CNN, and a detection head. At 0.5 IoU, we achieved mAP (mean average precision) of 0.6022 for car detection, 0.2313 for all objects detection and 0.2615 for three objects detection (car, pedestrian and cyclist). The third algorithm we've used is the feature pyramid network (FPN) with ResNet-18. Fundamentally the model consists of a backbone layer (e.g. resnet) and a feature pyramid network. For object detection, the CenterNet detection algorithm is used along with FPN. At 0.5 IoU, we achieved mAP of 0.97 for three objects detection under moderate difficulty. This

model showed that it is possible to use a single model to learn shared features and use multiple heads to perform various tasks. We also realized that it requires significant computational resources to work with lidar scans and due to that we had to make some compromises.

1. Introduction

Autonomous driving has been an active area of AI research in recent years. If the technology is commercialized successfully, it has the potential to greatly reduce traffic accidents, road congestion, and also the associated economic losses. The implementation of safe autonomous driving requires detection of surrounding obstacles, moving objects and also identifying drivable areas. Current autonomous vehicles use a mixture of radar, LIDAR, camera, high precision GPS, and prior map information to determine obstacles around the car and safe driving areas on the road surface. The functionalities of hardwares in self-driving cars are illustrated in figure 24 in the Appendix section[24]. In our final project, we aim to tackle the object detection problem associated with autonomous driving with three different types of algorithms: YOLO, PointPillars network and feature pyramid network (FPN) with ResNet as the backbone. The inputs to our algorithm are mainly two types of data. The first type of data is the group of snapshot images taken when the vehicle is driving down the road. The second type of data contains all the LIDAR data points collected by the LIDAR sensors mounted on the vehicle. There are other supplementary data, which we will discuss in more details in the "Dataset and Features" section. The outputs are groups of predicted 3D bounding volumes, which stand for the location, type and size of objects in the images. The final result is evaluated using mAP (mean average precision) at different IoU (intersection over union) thresholds.

2. Related Work

2.1. Object detection using CNNs

The first paper, Rich Feature Hierarchies for Accurate Object Detection and Semantic Representation, proposes a simple and scalable algorithm that combines two key insights: (1) one can apply high-capacity convolutional neural networks (CNNs) to bottom-up region proposals in order to localize and segment objects and (2) when labeled training data is scarce, supervised pre-training for an auxiliary task, followed by domain-specific fine-tuning, yields a performance boosts. The result is a combination of region proposals and CNN, called R-CNN [7]. The second paper, Mask R-CNN, proposes a method called Mask R-CNN. Mask R-CNN extends the R-CNN method proposed in the first paper by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression [8]. In the third paper, Frustum PointNets for 3D Object Detection from RGB-D Data, instead of solely relying on 3D proposals, the method of this paper leverages both mature 2D object detectors and advanced 3D deep learning for object localization, achieving efficiency as well as high recall from even small objects. [19]. The highest accuracy object detectors to date are based on a two-stage approach popularized by R-CNN. In contrast, one-stage detectors that are applied over a regular, dense sampling of possible object locations have the potential to be faster and simpler, but have trailed the accuracy of two-stage detectors thus far. The fourth paper, Focal Loss for Dense Object Detection, discovers that the extreme foreground-background class imbalance encountered during training of dense detectors is the central cause. The paper proposes to address this class imbalance by reshaping the standard cross entropy loss such that it down-weights the loss assigned to well-classified examples [12]. The fifth paper, Microsoft COCO: Common Objects in Context, presents a new dataset with the goal of advancing the state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding. This is achieved by gathering images of complex everyday scenes containing common objects in their natural context [13]. The sixth paper, Single-shot Multibox Detector, presents a method for detecting objects in images using a single deep neural network. This approach, named SSD, discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape [14].

The first paper [7] established that convolutional neural

network (CNN) architectures are state of art for detection in images. The second [8] and third paper [19] that followed it advocate a two-stage approach to this problem. In the first stage, a region proposal network suggests candidate proposals, which are cropped and resized before being classified by a second stage network. Both the fifth paper [13] and the sixth paper [14] have used the two-stage network. In a single-stage architecture, a dense set of anchor boxes is regressed and classified in one step into a set of predictions providing a fast and simple architecture. The fourth paper [12] convincingly argued that with their proposed focal loss function a single stage method is superior to two-stage methods, both in terms of accuracy and runtime.

2.2. Object detection in LIDAR point clouds

The sixth paper, Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks, proposes a computationally efficient approach to detecting objects natively in 3D point clouds using convolutional neural networks (CNNs). In particular, this is achieved by leveraging a feature-centric voting scheme to implement novel convolutional layers which explicitly exploit the sparsity encountered in the input [6]. The seventh paper, Multi-View 3D Object Detection Network for Autonomous Driving, proposes a Multi-View 3D networks (MV3D), a sensory-fusion framework that takes both LIDAR point cloud and RGB images as input and predicts oriented 3D bounding boxes and encodes the sparse 3D point cloud with a compact multi-view representation [3]. The eighth paper, PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, designs a novel type of neural network that directly consumes point clouds, which well respects the permutation invariance of points in the input. The network, named PointNet, provides a unified architecture for applications ranging from object classification, part segmentation, to scene semantic parsing [20]. The ninth paper, VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection, proposes VoxelNet, a generic 3D detection network that unifies feature extraction and bounding box prediction into a single stage, end-to-end trainable deep network. It divides a point cloud into equally spaced 3D voxels and transforms a group of points within each voxel into a unified feature representation through the newly introduced voxel feature encoding (VFE) layer [30]. The tenth paper, PointPillars: Fast Encoders for Object Detection from Point Clouds, proposes PointPillars, a novel encoder which utilizes PointNets to learn a representation of point clouds organized in vertical columns (pillars). While the encoded features can be used with any standard 2D convolutional detection architecture, this paper further propose a lean downstream network [10].

Object detection in point clouds is an intrinsically three dimensional problem. As such, it is natural to deploy a 3D convolutional network for detection, which is the paradigm of earlier work as in the sixth paper [6]. But these methods are slow. Most recent methods improve the runtime by projecting the 3D point cloud either onto the ground plane, as in the seventh paper[3]. In the most common paradigm the point cloud is organized in voxels and the set of voxels in each vertical column is encoded into a fixed-length, hand-crafted, feature encoding, such as MV3D[3], AVOD[9], PIXOR [28]and Complex YOLO[28], which all use variations on the same fixed encoding paradigm as the first step of their architectures. The eighth paper [20] proposed a simple architecture, PointNet, for learning from unordered point sets, which offered a path to full end-to-end learning. The ninth paper [30], which used VoxelNet, is one of the first methods to deploy PointNets for object detection in LIDAR point clouds. Here, PointNets are applied to voxels which are then processed by a set of 3D convolutional layers followed by a 2D backbone and a detection head. Frustum PointNet, as in the third paper [19], uses PointNets to segment and classify the point cloud in a frustum generated from projecting a detection on an image into 3D.

3. Methods

We have used three different algorithms here. The details of these algorithms are discussed below.

3.1. YOLO: 2D Object Detection

The first algorithm is YOLO, which stands for You Only Look Once. The YOLO family of models are a series of end-to-end deep learning models designed for fast object detection and is first described in the paper, You Only Look Once: Unified, Real-Time Object Detection [23]. The approach involves a single deep convolutional neural network that splits the input into a grid of cells and each cell directly predicts a bounding box and object classification. The result is a large number of candidate bounding boxes that are consolidated into a final prediction by a post-processing step [22]. There are three main variations of the approach: YOLOv1, YOLOv2, and YOLOv3. YOLOv1 proposed the general architecture. YOLOv2 refined the design and made use of predefined anchor boxes to improve bounding box proposal. YOLOv3 further refined the model architecture and training process [22]. In this project, we used the pre-trained weights of YOLOv3 system, called yolov3.weights [21]. These weights are trained using the DarkNet code base on the MSCOCO dataset. Then, we define a Keras model[4] that has the right number and type of layers to match the downloaded model weights. The

model architecture is called a “DarkNet” and was originally loosely based on the VGG-16 model. Then, we can use the model to directly detect the objects in the training set images/test set images.

3.2. PointPillars Network: 3D Object Detection

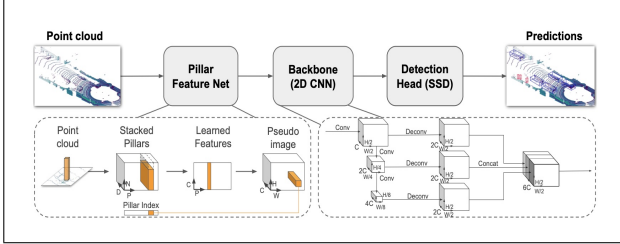
The second algorithm we’ve used is the PointPillars network. The architecture of the PointPillars network is in figure 2 below. PointPillars accept point clouds as input and estimates oriented 3D boxes. It has three main stages: (1) a feature encoder network that converts a point cloud to a sparse pseudo-image; (2) a 2D convolutional backbone to process the pseudo-image into high-level representation; and (3) a detection head that detects and regresses 3D boxes [10]. Here, we do a brief overview of PointPillar network. Please refer to the paper [10] for details. To apply a 2D convolutional architecture, we first convert the point cloud to a pseudo-image. We denote by l a point in a point cloud with coordinates x, y, z and the reflectance r . The point cloud is discretized into an evenly spaced grid in the $x - y$ plane, creating a set of pillars P with $|P| = B$. The points in each pillar are then augmented with x_c, y_c, z_c, x_p and y_p where the 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 [10].

The backbone has two sub-networks: one top-down network that produces features at increasingly small spatial resolution and a second network that performs upsampling and concatenation of the top-down features. The top-down backbone can be characterized by a series of blocks Block(S,L,F). Each block operates at stride S. A block has convolutional layers with F output channels, each followed by BatchNorm and a ReLU. The final features from each top-down block are combined through upsampling and concatenation. First, the features are upsampled from an initial stride to a final stride using a transposed 2D convolution with F final features. Next, BatchNorm and ReLU are applied to the upsampled features. The final output features are a concatenation of all features from different strides [10]. Finally, the Single Shot Detector (SSD) setup is used to perform 3D object detection. Similar to SSD, the priorboxes are matched to the ground truth using 2D IoU (intersection over union) [10]. Please see figure 20, 21, 22 and 23 in the Appendix section for four sample images that have bird’s-eye view of the lidar point cloud as well as the 3D bounding boxes projected into the image for clearer visualization. These sample images are from the paper[10]. We have built our implementations using code referenced in the Contributions and Acknowledgements section.

Rather than using any pretrained weights, we have

initialized the weights randomly using the uniform distribution. The encoder network uses 64 output features. The car and pedestrian backbones are the same except for the stride of the first block. All networks consist of three blocks. Each block is upsampled through the upsampling steps. We have used the same loss function as recommended in the paper [10]. Given ground truth boxes and anchors defined by $(x, y, z, w, l, h, \theta)$, the localization regression residuals between ground truth and anchors are defined by: $\Delta x = \frac{x^{gt} - x^a}{d^a}$, $\Delta y = \frac{y^{gt} - y^a}{d^a}$, $\Delta z = \frac{z^{gt} - z^a}{h^a}$, $\Delta w = \log \frac{w^{gt}}{w^a}$, $\Delta l = \log \frac{l^{gt}}{l^a}$, $\Delta h = \log \frac{h^{gt}}{h^a}$ and $\Delta \theta = \sin(\theta^{gt} - \theta^a)$. Here, x^{gt} and x^a are respectively the ground truth and anchor boxes and $d^a = \sqrt{(w^a)^2 + (l^a)^2}$. We have the total localization loss as $\mathcal{L}_{loc} = \sum_{b \in (x, y, z, w, l, h, \theta)} \text{SmoothL1}(\Delta b)$. Since the angle localization loss cannot distinguish flipped boxes, the heading is learned with a softmax classification loss, $\mathcal{L}_{dir}$, on the discretized directions. The object classification loss uses focal loss: $\mathcal{L}_{cls} = -\alpha_a(1 - p^a)^\gamma \log p^a$, where p^a is the class probability of an anchor. Using $\alpha = 0.25$ and $\gamma = 2$, the total loss is therefore: $\mathcal{L} = \frac{1}{N_{pos}}(\beta_{loc}\mathcal{L}_{loc} + \beta_{cls}\mathcal{L}_{cls} + \beta_{dir}\mathcal{L}_{dir})$, where N_{pos} is the number of positive anchors and $\beta_{loc} = 2$, $\beta_{cls} = 1$ and $\beta_{dir} = 0.2$. The loss function is optimized using Adam with an initial learning rate of 0.0002 which decays by a factor of 0.8 every 15 epochs.

Figure 1. PointPillars Network Overview (See Figure 19 in the Appendix for Larger Image) [10]



3.3. FPN: 3D Detection and Segmentation

For our 3D object detection and road segmentation task, we have used feature pyramid network (FPN) [11] with ResNet-18 as the backbone. This architecture is presented in Figure 2 below and augmented from the baseline model [5]. Fundamentally the model consists of a backbone layer (e.g. resnet-18 in this case) and a feature pyramid network. The feature pyramid network is nothing but a set of 1x1 conv layers with up-sampling to increase the spatial size of the final layers of backbone network. Skip connection is added between the backbone layers and Feature Pyramid layer to improve accuracy. For object detection, we've used CenterNet [29] detection algorithm. It is an NMS free and anchor free algorithm for object

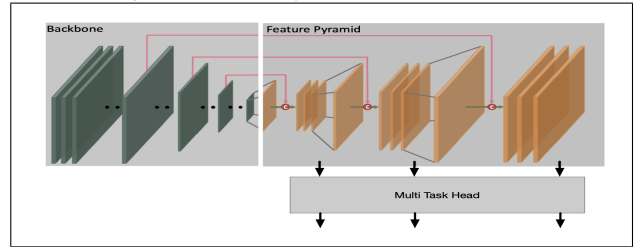
detection. CenterNet detects the center point of each object class in the image as heatmaps and uses regression loss to find the bounding box of the detected object. The model is multi-headed. We've used 6 heads in the model after the final layer to predict following outputs: heat-maps of the different classes present in the input, distance of the center of other objects from the self driving car, the angles in other objects are facing, dimensions of the detected objects, the z co-ordinate of the center of the object and the segmentation map.

Four types of loss functions are used in the model: (a) Focal loss [12], (b) L1 distance loss, (c) Balanced L1 loss and [17] (d) cross entropy loss [1]. Focal loss is used to encourage the network to learn correct classes.

$$focal_{loss} = -(1 - P_t)^r \log(P_t) \quad (1)$$

P_t is the confidence value of the network. $(1 - P_t)$ scale factor helps the network to focus more on the prediction it's less confident about. It is used to encourage the network to learn the correct classes from output heatmap. In short, the focal loss attempts to solve the class imbalance problem in dense single stage network where there is high amount of data in one class but less amount of data for the other class. L1 loss is used for the distance from ego car to the object and directions. Balanced L1 loss is used for the bounding box dimension and Z co-ordinate. We refer to [31] for a detailed description to the balanced L1 loss. In short, the multitask loss suffers from imbalances due to multiple loss functions and balanced L1 loss attempts to balance this so that the network converges. In this model both classification and localization are solved simultaneously in balanced L1 loss. Finally, cross entropy loss is used to do road segmentation. For learning rate scheduler, cosine annealing is used.

Figure 2. Feature Pyramid Architecture [11]



4. Dataset and Features

We have used two datasets in this project. Our initial plan was to use Waymo datasets [25][27][26]. But after considering the size of Waymo dataset (around 1.4TB) and the computational resources available to us, we have decided to find other datasets that are more suitable for the scope of

this project. These two datasets are Lyft Object Detection for Autonomous Vehicles Dataset and the 3D Object Detection dataset 2017 provided by the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. The details of these two datasets are discussed below.

4.1. YOLO and PointPillars Network

For YOLO and PointPillars network, the dataset we use is the Lyft Object Detection for Autonomous Vehicles Dataset [2]. The dataset contains LIDAR, image, map and data files for both train and test sets. The data files contain the primary identifier used for the training and test samples, including center x, center y, center z, width, length, height, yaw and class name. Center x, center y and center z are the world coordinates of the center of the 3D bounding volume. Width, length and height are the dimensions of the volume. Yaw is the angle of the volume around the z axis. Class name is the type of object contained by the bounding volume. Here, there are 10 types of objects, including animal, bicycle, bus, car, emergency vehicle, motorcycle, truck, pedestrian and other vehicle. Training and test set maps contain the maps of the sample area. The training image and test image files contain all the snapshot images captured by car sensors when driving down the road. We've included some sample images in the appendix section. The training image dataset is 23GB in size, and has 158,757 images. The test image dataset is 27.53 GB in size, and has 192,276 images. The LIDAR dataset contains all the LIDAR data points. The training LIDAR dataset is 39.56 GB in size and has 30,744 items. The test LIDAR dataset is 35.56 GB in size and has 27,468 items. However, due to limitations of computational resources and time, for the YOLO model and the PointPillars network, we have only used 25% of the overall training set and test set in the Lyft Object Detection for Autonomous Vehicles Dataset. For YOLO, we mainly used the image data. For PointPillars network, we mainly used the LIDAR data.

4.2. FPN: 3D Detection and Segmentation

For FPN, the dataset we use is the 3D Object Detection dataset 2017 provided by the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago[16]. In this dataset, the 3D object detection benchmark consists of 7,481 training images and 7,518 test images as well as the corresponding point clouds, comprising a total of 80,256 labeled objects. We used the point cloud only for the detection task. It includes Velodyne point cloud data and training labels 15,000 point clouds. The training data is split into 3000/1000 training and validation sets. The point cloud data is first projected into a bird eye view plane by using the camera and lidar calibration data provided with the dataset. The size of resulting bird eye view frame is

608x608. This bird eye view frame is used as an input to the model. Finally we used KITTI Road/Lane Detection Evaluation 2013 [15] dataset for the road segmentation task. We had to find another dataset for segmentation because the 3d detection dataset mentioned above didn't contain any labels for segmentation. The dataset contains 100 images and manually annotated segmentation mask for road and lanes. The dataset is split into 90/10 images of training and validation set. The original image size is (160, 576). However the images are reshaped into 152x608 during pre-processing step. At the input of the model the image is interpolated from 152x608 to 608x608 to make it compatible with the 3D detection pipeline as they both share the same architecture. For segmentation dataset, no data augmentation is performed but for the detection dataset, random horizontal flips are used to augment the data. Figure 3 below depicts a visualization of point cloud data from KITTI training dataset performed using open3D library. The KITTI dataset provided point cloud in .bin format. We first converted it to .pcd format for this visualization purpose. Figure 4 shows the sample training data and labels for segmentation dataset.

Figure 3. Example point cloud data

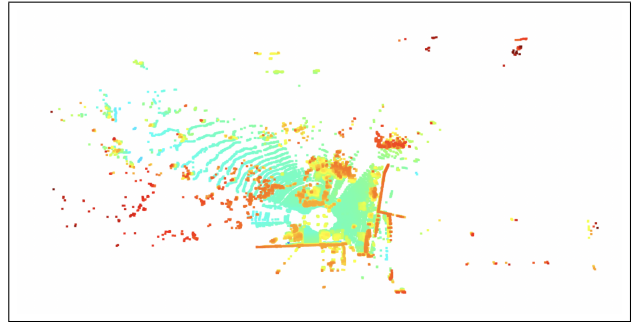
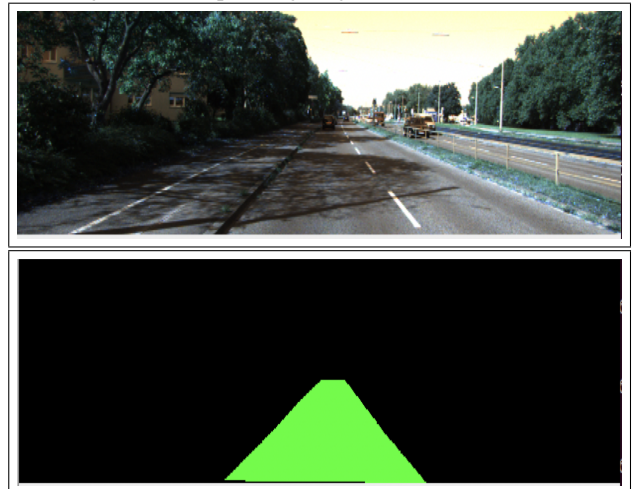


Figure 4. Example image segmentation data for lane



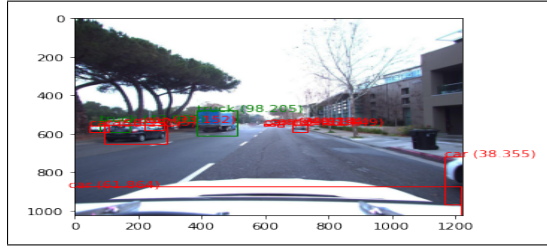
5. Experiments/Results/Discussion

To train and test the model in this final project, we have used a combination of Google Colaboratory, Amazon Web Services and NVIDIA GeForce RTX 3080 Ti.

5.1. YOLO: 2D Object Detection

Applying the YOLO model on the training set images from Lyft Object Detection for Autonomous Vehicles Dataset, we get a group of sample detection images on the training set. Sample image 1 is shown below. Other sample images (2 and 3) are shown in the Appendix section. Since the YOLO model has used the pre-trained weights and doesn't involve any training, we have only used it for doing qualitative analysis. After going through these images, we can see that the YOLO model can correctly identify cars with red bounding box and trucks with green bounding boxes like below in many cases. This indicates that the YOLO model works in many cases. For implementation, we mainly used the Keras of TensorFlow library here [4].

Figure 5. Sample Image 1 Using the YOLO model



5.2. PointPillars Network: 3D Object Detection

For the PointPillars Network, rather than using any pre-trained weights, we have initialized the weights randomly using the uniform distribution. We have used the same loss function and hyperparameters as recommended in the paper [10]. The loss function is optimized using Adam with an initial learning rate of 0.0002 which decays by a factor of 0.8 every 15 epochs. The architectures of the PointPillars network are discussed in details in the Methods section. The model is trained for 200 epochs. For implementation, we mainly used the PyTorch library here [18]. We evaluate the PointPillars network on the mAP (mean average precision) at different IoU (intersection over union) thresholds. The IoU of a set of predicted 3D bounding volumes and ground truth bounding volumes is calculated as: $IoU = \frac{A \cap B}{A \cup B}$. The metric sweeps over a range of IoU thresholds, at each point calculating an average precision value. The threshold values range from 0.5 to 0.95 with a step size of 0.05: (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95). At each threshold value t , a precision value is calculated based on the number of true positives (TP), false negatives (FN), and false positives (FP) resulting from comparing

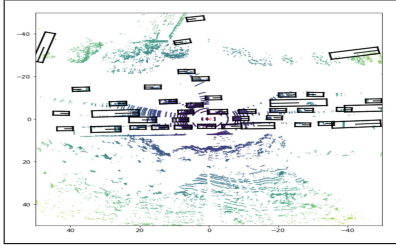
the predicted object to all ground truth objects. A true positive is counted when a single predicted object matches a ground truth object with an IoU above the threshold. A false positive indicates a predicted object has no associated ground truth object. A false negative indicates a ground truth object has no associated predicted object. The average precision of a single image is calculated as the mean of the above precision values at each IoU threshold: $\frac{1}{|thresholds|} \sum_t \frac{TP(t)}{TP(t) + FP(t) + FN(t)}$. Using the Lyft Object Detection for Autonomous Vehicles Dataset, we get the following table, which contains the mAP per category per IoU threshold.

Figure 6. Mean Average Precision Per Category Per IoU Threshold

IOU threshold	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
Animal	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Bicycle	0.1096	0.0953	0.0817	0.0686	0.0571	0.0481	0.0412	0.0361	0.0321	0.0289
Bus	0.3925	0.3873	0.3794	0.3669	0.3500	0.3242	0.2909	0.2569	0.2285	0.2056
Car	0.6022	0.5866	0.5644	0.5331	0.4914	0.4407	0.3880	0.3410	0.3032	0.2729
Emergency Vehicle	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Motorcycle	0.0936	0.0716	0.0538	0.0419	0.0340	0.0284	0.0244	0.0213	0.0190	0.0171
Other Vehicle	0.4316	0.4226	0.4112	0.3957	0.3747	0.3464	0.3107	0.2743	0.2439	0.2195
Pedestrian	0.0728	0.0569	0.0440	0.0343	0.0277	0.0231	0.0198	0.0173	0.0154	0.0139
Truck	0.3798	0.3717	0.3606	0.3452	0.3258	0.2995	0.2686	0.2375	0.2113	0.1902
Average Over Car, Pedestrian and Bicycle/Cyclist	0.2615	0.2463	0.2300	0.2120	0.1921	0.1706	0.1497	0.1315	0.1169	0.1052
Average Over All	0.2313	0.2213	0.2106	0.1984	0.1845	0.1678	0.1493	0.1316	0.1170	0.1053

From the table above, we conclude the following. First, among all the categories, car, bus, truck and other vehicle have the highest mAP (mean average precision). Animal and emergency vehicle have zero mAP, which is likely because there are very few animals and emergency vehicle in the training and test set images. Bicycles and pedestrians have low mAP. Second, when we increase the IoU threshold values, the mAP values decrease in all categories. Third, from the PointPillars paper [10], we can see that the mAP value for detecting cars can reach the level of 0.7 to 0.9 at 0.5 IoU. However, at 0.5 IoU, our mAP value for car detection is only around 0.6022. Since we are using the same architecture as the one in the paper, we think there are two possible reasons for the underperformance. The first possible reason is that the paper uses KITTI dataset but we are using the LFYT dataset. It's possible that the KITTI dataset contains less noise than the LYFT dataset. The second possible reason is that we have only trained and tested on 25% of the Lyft Object Detection for Autonomous Vehicles Dataset. If we can train and test on the full dataset, then our mAP might be higher. Fourth, bicycle and pedestrian have low mAP. After going through some sample images, we find that one common mistake is that the two can be misclassified with each other. They can also be confused with other narrow objects in the image, such as trees or poles. We have plotted predicted and ground truth boxes on corresponding LIDAR point cloud pred boxes in LIDAR's field of regard. The sample plot is as follows.

Figure 7. Plot of predicted and ground truth boxes on corresponding LIDAR point cloud pred boxes in LIDAR's field of regard



5.3. 3D Detection and Road Segmentation

For the 3D detection and road segmentation task, we augmented the baseline model by adding new layers and infrastructure code to perform multi-task detection and segmentation using the same model but with different heads for different tasks. For detection we used mAP (mean average precision) metric to evaluate the model. We used 0.001 learning rate with cosine learning rate scheduler with Adam SGD optimizer and batch size 16. To simplify the project, we only attempted to detect 3 categories - car, pedestrian and cyclists, and for segmentation task, only detected road and lanes. The KITTI detection dataset didn't come with any segmentation mask. So we had to find an auxiliary dataset just for the segmentation task as discussed above in the dataset section. Following table shows the mAP achieved for the detection tasks. The KITTI dataset is stratified into easy, moderate, and hard difficulties. Please note that we are not reporting accuracy for the segmentation tasks. The following table only reports results for 3d detection and resulting 2d bounding box derived from lidar 3d bounding boxes. We've created two demonstration videos using the model predictions, one on detection and one on segmentation. It can be accessed in here: <https://drive.google.com/drive/folders/1duKIdffuq-9HpZGZhiYpAf7KSZmYlT8m?usp=sharing>

Figure 8. mAP for 3D Detection Tasks (Average Over Car, Pedestrian and Cyclist)

Task	Hard	Moderate	Easy	Num recall	Num epochs
bbox	30.04	33.28	37.45	11	10
3d	12.13	12.30	13.99	11	10
bbox	30.29	34.32	36.05	40	10
3d	12.79	12.85	13.08	40	10
bbox	72	78	80	11	300
3d	95	97	97	11	300
bbox	72	84	87	40	300
3d	97	97	98	40	300

We reported the results after training for 10 epochs and 300 epochs. The model took approximately 15 hours for 300 epochs and to achieve good results it was necessary to train it for longer period of time as shown in the table above. Following figures shows the outputs from the model with 3d

bounding boxes and road/lane segmentation.

Figure 9. Output from model- 3d Detection. Bottom image shows the bird eye view of lidar image and the top image shows the camera image (See figure20 in Appendix for larger regular-size image)

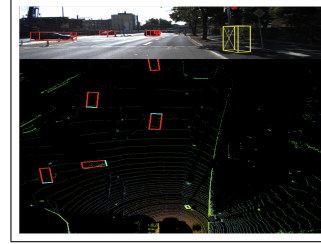
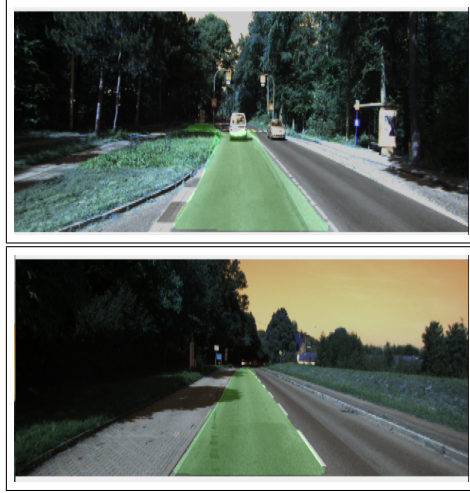


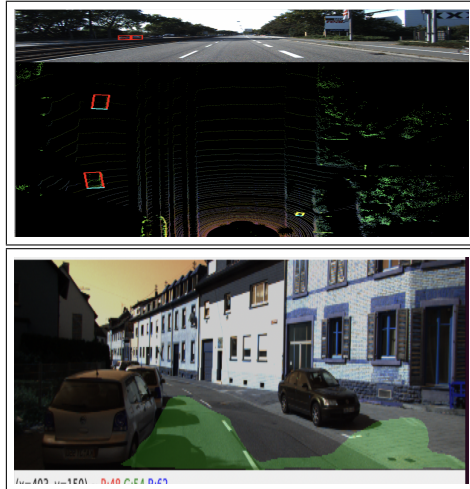
Figure 10. Output from model- road/lane segmentation



5.4. Failure Modes

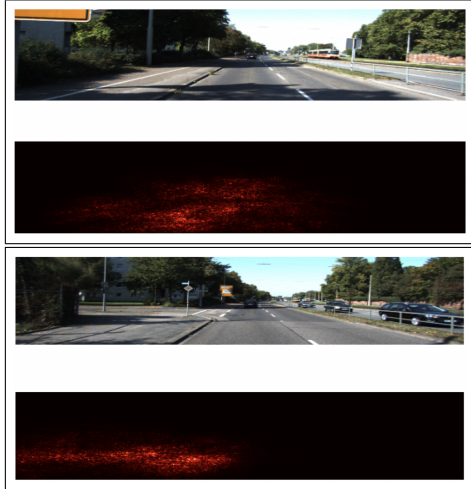
Even though visually we were able to achieved good results, the model failed in some edge cases. Following images shows some example of bad predictions.

Figure 11. Model - failure modes (after training for 50 epochs)



For the detection task, the model fails to detect distant object or object in the periphery. For the segmentation task, it performs poorly when there is more than 2 lanes or curved road with blind spots as shown above. We were able to alleviate the detection errors shown above just by training the model for longer. For example we noticed that the model didn't suffer with similar detection errors after training for 300 epochs. As the 3d detection is performed using point cloud data, we hypothesize that we just need to train the model for longer period to achieve better performance. Another thing to note is that we have only used 3000 lidar scans to train the model to cut down training time. It also adversely affected the performance. However for the segmentation task, training the model for longer didn't alleviate the issues. To understand what the model is looking at we've generated some saliency maps as shown below.

Figure 12. Saliency maps



The top image in figure 13 shows that the model is paying more attention to the lane as expected. The bottom image shows that the model is not properly detecting the lane if there is a left turn ahead. As training the model for more epochs didn't alleviate the issues, the model is most likely under-fitting. We had to add some interpolation layers to up-sample from 152 to 608 in horizontal dimension before feeding the image to the model. We had to do it so that the same architecture can be used for both detection and segmentation. Due to time constraint we were not able to do any further exploration but we suspect that the model is not performing well due to this. Another important factor is the lack of training data. We strongly believe that if we train the model with more image segmentation data then the performance will improve significantly. We only had 100 images in this segmentation dataset.

6. Conclusion/Future Work

For our project, we have mainly used three algorithms for object detection. The first algorithm is YOLO, which stands for You Only Look Once. YOLO is a 2D object detection algorithm, and its YOLOv3 system gives us the pre-trained weights that can be used in object detection. Since we don't need to do any training here, we have mainly used the YOLO algorithm for qualitative analysis. After going through these images, we can see that the YOLO model can correctly identify cars with red bounding box and trucks with green bounding boxes in many images, which indicates that the YOLO model works in many cases.

The second algorithm is the PointPillars network, which is a novel encoder that utilizes PointNets to learn a representation of point clouds organized in vertical columns (pillars). The overall architecture of PointPillars contains three components: a pillar feature net, a backbone 2D CNN, and a detection head. Using the PointPillars network, we are able to achieve a mean average prediction of 0.6022 at 0.5 IoU for car detection. Our PointPillars network has underperformed the one in the PointPillars paper[10], which has a mean average precision value range of 0.7 to 0.9 at 0.5 IoU for car detection. Since we are using the same architecture as the one in the paper, we think there are two possible reasons for the underperformance. The first possible reason is that the paper uses KITTI dataset but we are using the LYFT dataset. It's possible that the KITTI dataset contains less noise than the LYFT dataset. The second possible reason is that we have only trained and tested on 25% of the Lyft Object Detection for Autonomous Vehicles Dataset. If we can train and test on the full dataset, then our mAP might be higher. So, for our future work, if we have more computational resources and time, we should try to train and test on the full dataset and try to get the most optimal performance. We should also tune the hyperparameters more to further optimize the performances.

Finally, we have experimented with multi-task learning using keypoint detection and feature pyramid architecture [11]. The model showed that it is possible to use a shared model and add multiple heads to perform various tasks. We also realized that it requires significant computational resources to work with lidar data and in future we would like to run more experiments with various hyperparameters and model configurations to improve the overall result.

7. Appendices (optional and not part of page limit)

Figure 13. Sample Image 1 in the Lyft Object Detection for Autonomous Vehicles Dataset



Figure 14. Sample Image 2 in the Lyft Object Detection for Autonomous Vehicles Dataset



Figure 15. Sample Image 3 in the Lyft Object Detection for Autonomous Vehicles Dataset



Figure 16. Sample Image 2 Using the YOLO model

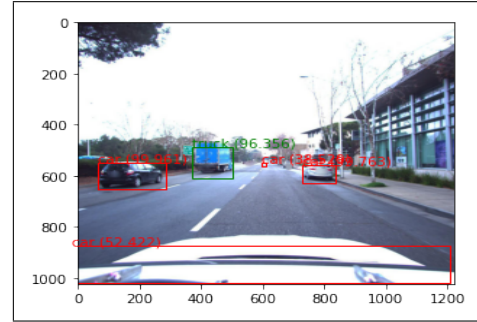


Figure 17. Sample Image 3 Using the YOLO model

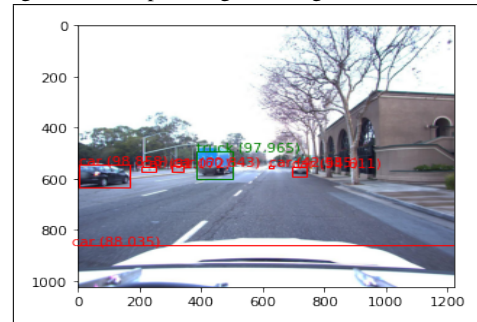


Figure 18. PointPillars Network Overview (Larger) [10]

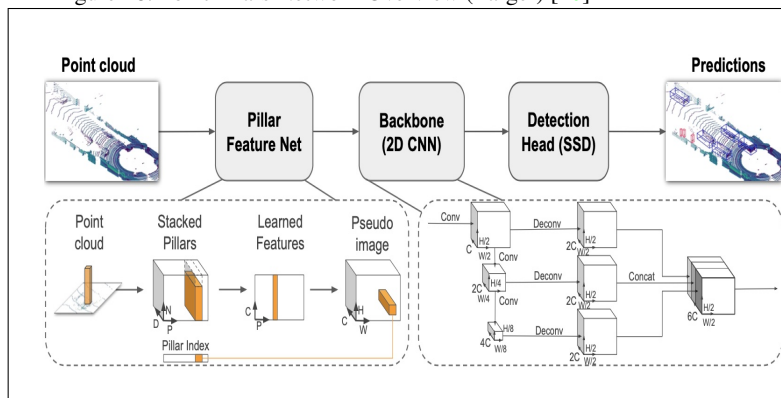


Figure 19. Output from model- 3d Detection. Bottom image shows the bird eye view of lidar image and the top image shows the camera image (Larger Image)

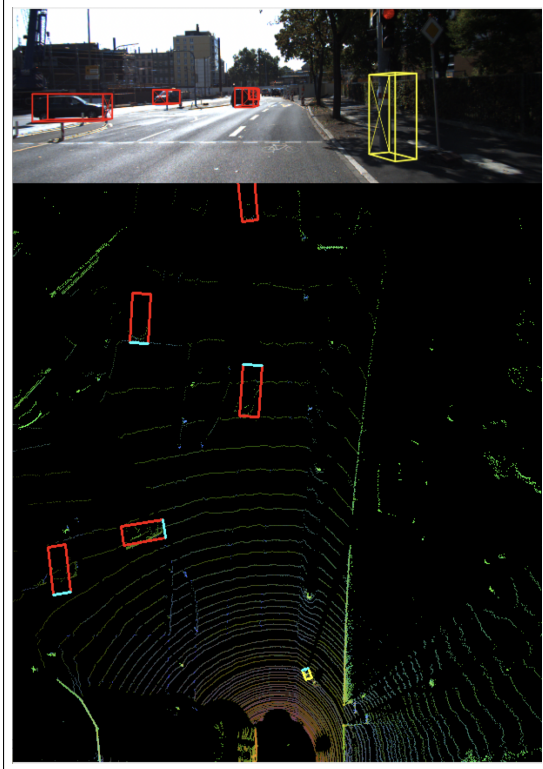


Figure 20. Sample Image 1 - a bird's-eye view of the lidar point cloud (top), as well as the 3D bounding boxes projected into the image for clearer visualization. The predicted boxes for car is orange), for cyclist is red and for pedestrian is blue. Ground truth boxes are shown in gray. The orientation of boxes is shown by a line connected the bottom center to the front of the box. [10]

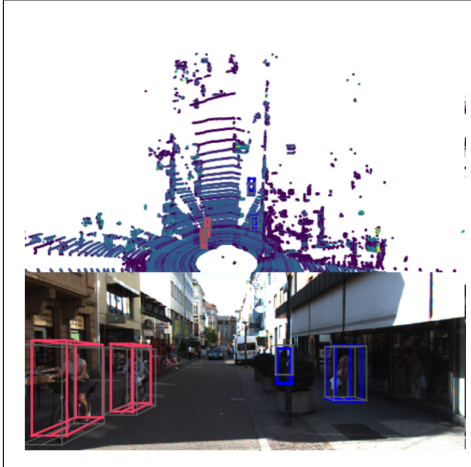


Figure 21. Sample image 2 - a bird's-eye view of the lidar point cloud (top), as well as the 3D bounding boxes projected into the image for clearer visualization. The predicted boxes for car is orange), for cyclist is red and for pedestrian is blue. Ground truth boxes are shown in gray. The orientation of boxes is shown by a line connected the bottom center to the front of the box. [10]

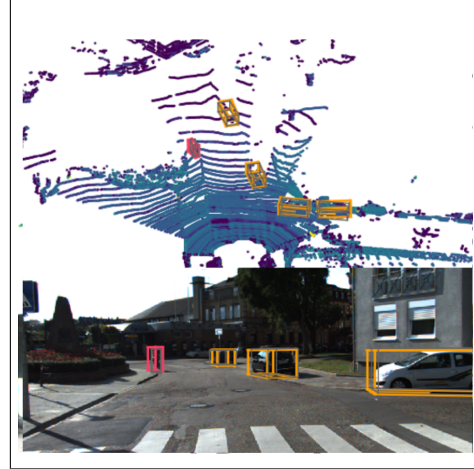


Figure 22. Sample image 3 - a bird's-eye view of the lidar point cloud (top), as well as the 3D bounding boxes projected into the image for clearer visualization. The predicted boxes for car is orange), for cyclist is red and for pedestrian is blue. Ground truth boxes are shown in gray. The orientation of boxes is shown by a line connected the bottom center to the front of the box.[10]

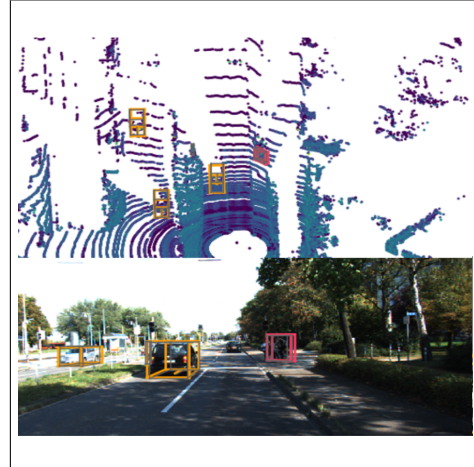


Figure 23. Sample image 4 - a bird's-eye view of the lidar point cloud (top), as well as the 3D bounding boxes projected into the image for clearer visualization. The predicted boxes for car is orange), for cyclist is red and for pedestrian is blue. Ground truth boxes are shown in gray. The orientation of boxes is shown by a line connected the bottom center to the front of the box.[10]

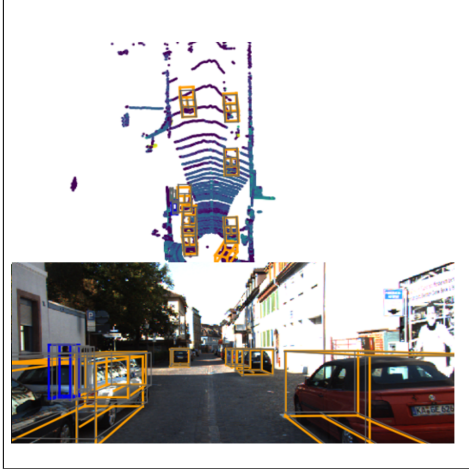
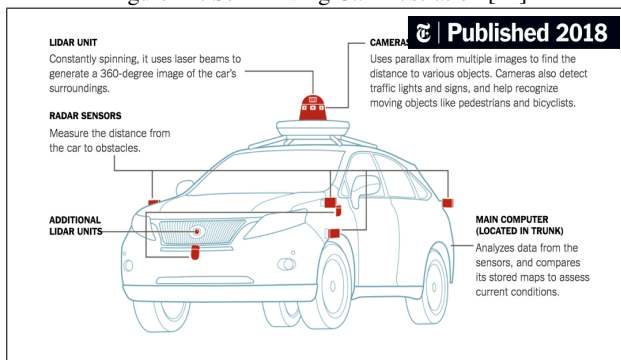


Figure 24. Self Driving Car Illustration [24]



8. Contributions and Acknowledgements (not part of page limit)

8.1. Contributions

Both Adil Sadik and Qianli Song have contributed equally to the project. Unfortunately, Rui Chen was busy with his job and so was unable to contribute significantly to the final project.

- Qianli Song worked on the YOLO model and PointPillars network model using the Lyft Object Detection for Autonomous Vehicles Dataset.

- Adil Sadik worked on the 3D detection and road/lane segmentation tasks using KITTI datasets.

8.2. Acknowledgements

Due to the inherent complexities of this project topic, we have built our project on a number of third-party resources and codes.

- For YOLO model, we have built our projects based on code from this link (<https://github.com/keshik6/KITTI-2d-object-detection>), resource from this website (<https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/>) and resource from this website (<https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/>).

- For PointPillars network model, we have built our projects based on code from this link (<https://github.com/traveller59/second.pytorch>), resource from this website (<https://becominghuman.ai/train-a-3-d-object-detection-model-point-pillars-in-a-jupyter-notebook-1911ddd5bfe4>) and also resource from this website (<https://towardsdatascience.com/implementing-point-pillars-in-tensorflow-c38d10e9286>).

- For feature pyramid network (FPN), we've used (<https://github.com/maudzung/SFA3D>) as the baseline model and augmented it to fit our purpose.

9. References/Bibliography (No page limit)

References

- [1] 4
- [2] LYFT Level 5. LYFT Level 5 Object Detection Dataset. <https://level-5.global/data/>, 2019. [Online; accessed 29-May-2022]. 5
- [3] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving, 2016. 2, 3
- [4] François Chollet. Keras TensorFlow. <https://keras.io/>, 2015. [Online; accessed 27-May-2022]. 3, 6
- [5] Nguyen Mau Dzong. Super fast and accurate 3d object detection based on 3d lidar point clouds: <https://github.com/maudzung/sfa3d>. 4
- [6] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks, 2016. 2, 3

- [7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013. 2
- [8] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2017. 2
- [9] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. Joint 3d proposal generation and object detection from view aggregation, 2017. 3
- [10] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds, 2018. 2, 3, 4, 6, 8, 9, 10, 11
- [11] Peixuan Li, Huaici Zhao, Pengfei Liu, and Feidao Cao. Rtm3d: Real-time monocular 3d detection from object keypoints for autonomous driving, 2020. 4, 8
- [12] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2017. 2, 4
- [13] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014. 2
- [14] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016. 2
- [15] Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. 5
- [16] Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. 3d object detection evaluation 2017 provided by karlsruhe institute of technology and toyota technological institute at chicago: http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d. 5
- [17] Jiangmiao Pang, Kai Chen, Jianping Shi, Huajun Feng, Wanli Ouyang, and Dahua Lin. Libra R-CNN: towards balanced learning for object detection. *CoRR*, abs/1904.02701, 2019. 4
- [18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 6
- [19] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from rgb-d data, 2017. 2, 3
- [20] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2016. 2, 3
- [21] Joseph Redmon. YOLOv3 Pre-trained Weights. <https://pjreddie.com/darknet/yolo/>, 2015. [Online; accessed 29-May-2022]. 3
- [22] Joseph Redmon. How to Perform Object Detection With YOLOv3 in Keras. <https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/>, 2019. [Online; accessed 29-May-2022]. 3
- [23] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015. 3
- [24] New York Times. How Driverless Cars See the World Around Them. <https://www.nytimes.com/2018/03/19/technology/how-driverless-cars-work.html>, 2018. [Online; accessed 31-May-2022]. 1, 11
- [25] Waymo. Waymo Open Dataset Motion. <https://waymo.com/open/data/motion/>, 2021. [Online; accessed 29-May-2022]. 4
- [26] Waymo. Waymo Open Dataset Overview. https://waymo.com/intl/en_us/open/challenges/, 2021. [Online; accessed 29-May-2022]. 4
- [27] Waymo. Waymo Open Dataset Perception. <https://waymo.com/open/data/perception>, 2021. [Online; accessed 29-May-2022]. 4
- [28] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds, 2019. 3
- [29] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv preprint arXiv:1904.07850*, 2019. 4
- [30] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection, 2017. 2, 3
- [31] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *arXiv preprint arXiv:1908.09492*, 2019. 4