Hochschule Ravensburg-Weingarten

Lidar and Radar Systems: Project Report 2

YOLO Object Detector Evaluation on Astyx HiRes 2019 Dataset

Created by:

Sahil Arora
(Matriculation Number: 32953)

January 22, 2021

Contents

1 Introduction

When we think of an autonomous application such as a self-driving car, a wide range of sensors come to mind that can be employed to do the job. On one side, you may have seen prototypes of Autonomous Taxis by Waymo with a large black colored protruding LiDAR sensor on top of it [Waya]. LiDAR sensor emits laser beams which when strikes an obstacle, reflect back to sensor. The time elapsed can be used to measure the obstacles' distance [Wika]. A critical issue in development of autonomous vehicles is adverse weather conditions. Current Vision and LiDAR sensors may not work at best potential during challenging conditions like fog, snow etc [Zan+19]. Here comes another sensor that is historically used in military applications. RADAR is now also being used heavily in automobile industry to serve as an assistance device to driver. It works by emitting Electromagnetic Waves and sensing the reflected ones to compute the time of flight and properties like distance, velocity of the obstacle it hit [Wikb].

A lot of companies are investing resources in training and testing, thereby publishing a large amount of datasets openly available for the researchers [Cae+20]. One of these is Astyx HiRes 2019 dataset which is what we use in this project [MK19]. The scope of this project is to evaluate already trained Object Detectors for LiDAR and RADAR data. Section 2 deals with loading the Object Detector and the dataset used to validate the Detectors. Section 3, deals with using LiDAR data to predict bounding boxes and computing the Intersection-over-Union value for the detected objects w.r.t. corresponding Ground Truth data, if present. Average Precision is computed to define the working efficiency of our model. Section 4 contains the same process on RADAR data, highlighting a few important notes. Section 5 provides examples where the Detector performed well and where it failed.

The project requires a few tools and libraries which need to be set up. The estimations and evaluations are performed on Ubuntu 18.4 platform running on a Intel i7 personal computer. The code can be referred at Github repository [Git] Source code is written in Python with support libraries like

- NumPy [Num]
- MatPlotLib [M D]
- Shapely [Sha]
- PyTorch [PyT]
- OS [OSP]

2 Object Detector and Dataset

2.1 YOLO Object Detection Algorithm

We are using a Complex YOLO object detection algorithm, which is a popular choice for detection because of its high accuracy and fast processing time. The algorithms required only one forward propagation through a Convolutional Neural Network to make predictions. Hence one can say, it only looks once. The CNN is then able to predict bounding boxes and probability of the class of each bounding box [Red+16].

2.2 Darknet Convolutional Neural Network

The CNN which is used for this project is Darknet [R16]. Darknet is a Neural Network framework which is fast and accurate to detect objects in real time. It is an open source project and can be easily installed and coupled with YOLO algorithm, given is the guideline to do so at the homepage [R16]. But our focus of the task is not to train the Neural Network in this case. Rather, as problem statement, we are provided with the trained parameters of Darknet Neural Network with output weight vectors as *.pth* files. There are two separate weight files for the LiDAR data and the RADAR data. These files can just be plugged into the Darknet framework using PyTorch and the setup is ready to make predictions or for evaluation, whatever is needed.

2.3 Astyx HiRes 2019 Dataset

The model is trained on the Astyx HiRes 2019 Dataset (which can be downloaded from the website) [Gmb]. There are a total of 546 scenes available in this dataset. Usually, to avoid Overfitting, a model is not trained on all of the given scenes, rather the given dataset is split into two parts: Training Set and Validation Set [Die95]. Keeping the training to validating ratio of 80 : 20 is a common practice as it reserves enough data to validate the model. In our case, since the model is already trained, we require just the Validation Set to evaluate the model. The ID of the scenes used for validation are provided in form a text file, which can be used to evaluate scene-by-scene. A total of 107 Scene IDs are given in the Validation File. Fig 2.3 shows one example of Camera Image and LiDAR observations mapped in the 2D image plane, corresponding to Scene ID 208.

Fig 2.3 Camera Image and 2D mapped LiDAR readings (Scene 208) [MK19] [Git]

3 Detections and Evaluation of LiDAR dataset

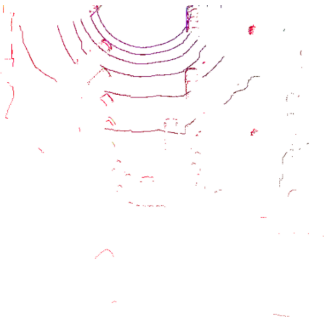
3.1 Transforming LiDAR data

The Astyx dataset is majorly divided into 5 different directories that can be accessed with help of python *os* library. The dataset contains Ground Truth, Camera Views, Lidar and Radar recordings and their calibration data [MK19]. The LiDAR point set is in a different co-ordinate system than RADAR point set. To visualize everything in a base frame (in this case RADAR frame), we need to transform the LiDAR point set with co-ordinate transformation matrix. These are provided to us by Astyx in the calibration directory for each individual Scene ID. We define a python class program to open the corresponding file w.r.t. given Scene ID and read the LiDAR-to-reference matrix from the calibration file. Then using the homogeneous coordinate transformation of given LiDAR points $(x, y, z) \rightarrow (x, y, z, 1)$, we multiply the homogenized co-ordinates with the LiDAR-to-reference matrix and get the transformed points [BR20].

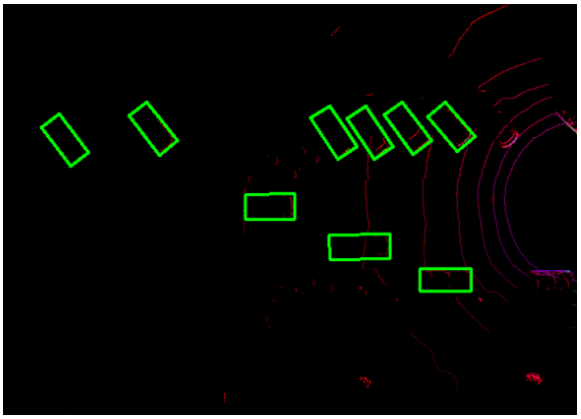
3.2 Bird's Eye View Transformation

3D is relatively easier for a human being to understand and interpret. For computers, it may not not be an easy job. So it is a good idea to convert 3D point cloud in 2D data which can be plotted on an image. We convert it to the BEV (Bird's Eye View). As the name suggests, it is an elevated view of an object from above, similar to what a bird would see from sky [Wikc]. Figure 3.2A shows the 3D LiDAR observations converted into a 2D image in BEV.

The Ground Truth data corresponding to every Scene ID is also provided to us in the Astyx dataset. This information is given in the format of a *.json* file and contains the information of the length, breadth and height of the object. Since we are dealing with BEV (in simpler terms, the top view), we can use the length and breadth dimensions to plot the corner points in the 2D image and use OpenCV function to draw lines along these points. This would give us the True Bounding Boxes for all objects present in that scene, which can be further used to compare and evaluate Detected Objects. Fig 3.2B shows Bounding Boxes (green colored) of Ground Truth objects in Scene 208.



(a) Fig 3.2A LiDAR in BEV (Scene 208) [MK19]



(b) Fig 3.2B Ground Truth (Scene 208)[MK19]

3.3 Detection by Darknet

For the detection algorithm, each Scene ID from the validation split is processed one by one. The image is first passed to the Darknet CNN which predicts the class probabilities of all the object found in the image. In our case, we have to deal with just one object i.e. Cars. The probabilities of object would range from 0 to 1 but to improve the efficiency of the model, we would like to consider only the predictions above a certain threshold. Therefore, we apply a Non-Maximum Suppression algorithm [NV06] and simultaneously considering only the prediction above a 95 % confidence interval. As it may happen in case of object detection that multiple bounding boxes are detected for the same object having different value of class probability, we want to choose the one with highest probability to best define the object. First of all, any bounding box with prediction score of <95% is simply rejected. Next, if any bounding boxes are still remaining to this corresponding image, then Intersection-over-Union [Rez+19] is computed for one of these boxes w.r.t to remaining eligible boxes. If the IoU comes out to be greater than a Non-Max Suppression threshold value of 40%, then box is retained, otherwise removed from the list. From this list of large retained IoU boxes, if their labels are matching, then these boxes are overlapped by order of their confidence. This process of done iteratively, until all bounding boxes are either thrown away or merged with others.

Definition of IoU: Intersection over Union is one of the evaluation metrics used to measure the correctness of a detected object on a given true data. It is also often referred as Jaccard Index [Ind], measuring the similarity between two observations. To compute the IoU value, we typically need two things:

1. Ground Truth bounding box dimensions, let's say **A**
2. Predicted BB dimensions by the Object Detector, let's say **B**

$$IoU = \frac{A \cap B}{A \cup B}$$

Using Python library *Shapely* [Sha] it is relatively easy to compute the IoU. Fig 3.3A shows the code used in this project to compute IoU.

```
def compute_iou(box, boxes):
    """Calculates IoU of the given box with the array of the given boxes.
    The IoU is the fraction between the area of the intersection and the union
        the bounding boxes.
    Input:
    box: a shapley polygon
    boxes: a list of shapley polygons
    Output:
    numpy array of all iou values between box and the boxes in boxes (dtype float32)
    """

    # Intitiate an empty array
    iou = [None] * len(boxes)

    # Run Loop for all boxes in range
    for p in range(len(boxes)):
        intersc = box.intersection(boxes[p]) # Insections using Shapely
        union = box.union(boxes[p]) # Union using Shapely

        iou[p] = intersc.area/union.area # Compute IoU

    iou = np.array(iou) # Convert to numpy array
    return iou
```

Fig 3.3A Source Code using Shapely [Git]

The output of the Detector is then stored as image for visualization. Each output image contains the plotted LiDAR observation in BEV, Bounding Boxes for the Ground Truth Object (shown in green rectangles) and Bounding Box for detected Objects (shown in red colored rectangle). Fig 3.3B shows one such detected example from Scene ID 208 of Astyx Dataset.

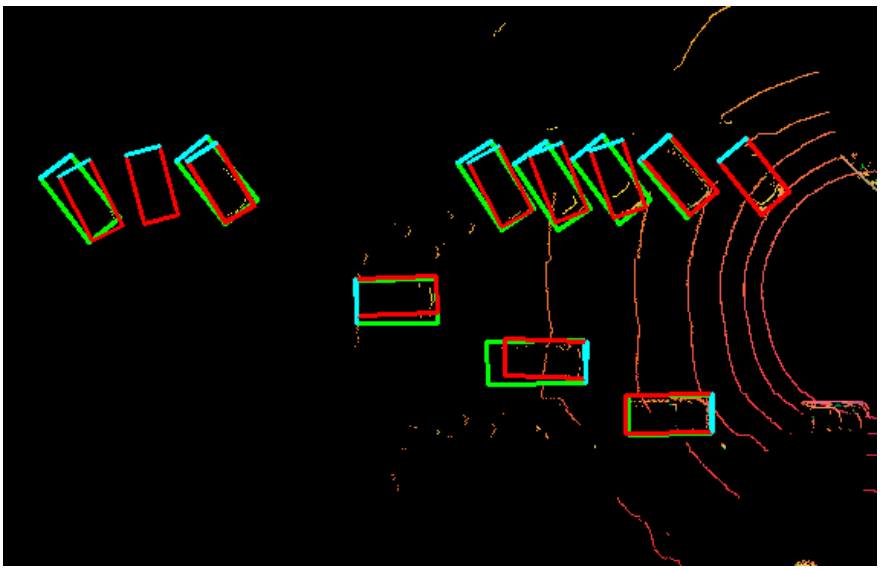


Fig 3.3B Detected and Ground Truth Objects in LiDAR BEV (Scene 208) [MK19] [Git]

3.4 Average Precision

Average Precision is a common metric used to measure the performance of an Object Detector [ZZ09]. To precisely define, what Average Precision means, we first have to define some common parameters:

The positive detections given by an Object Detector may be True (there is actually a Ground Truth associated with prediction) or False (there is no Ground Truth corresponding to this prediction). We use the IoU values to classify the predictions as **True Positive(TP)** or **False Positive(FP)** [ER20]. In our case we are using a value of 0.5, that means an IoU of more than 0.5 renders that detection as TP. Precision tells how accurate the predictions are. With the above two value we can compute the Precision of the model.

$$Precision = \frac{TP}{TP + FP}$$

On the the other hand, Recall is just the opposite of Precision, as it estimates the False Negative Rate of model. For each image, there can be **True Negative(TN)** and **False Negative(FN)** [ER20]. Since every part where the model didn‘t predict an object is negative, computing True Negatives is a futile job. We limit our scope to False Negatives, ie the objects that our model has failed to detect.

$$Recall = \frac{TP}{TP + FN}$$

Average Precision:

As the output of the detection algorithm, we have the prediction list with values of class probability. We compute a *numpy* array of size **m*n** where **m** is the number of predicted objects and **n** is the number of Ground Truth objects. The elements of this array are IoU values between i^{th} and j^{th} element of predicted and Ground Truth object lists respectively. By this method, we define True Positives (threshold value of 0.5) and the ones which are not True Positives are termed as False Positive. Now, we can compute the Precision and Recall with above given formulas. We compute the values cumulatively and store them as a list. This means, we take only 1 element from TP and FP list and compute Precision, then in next step include one more element from the list and again compute Precision. A same procedure is followed to compute the Recall, as done in [BR20]. This way we get a set of values for Precision and Recall, length of the set being count of Ground Truth objects.

11 - Point Average Precision:

To compute the 11 point Average Precision, the mean value of corresponding interpolated Precision values at a set of equally spaced 11 Recall values in the range of [0,1] is calculated [BR20] [ZZ09].

This means that values are interpolated for Precision keeping:

$$Recall \in [0, 0.1, 0.2, 0.3, ..., 0.8, 0.9, 1.0]$$

3.5 Result

Following are the results of LiDAR validation data.

Average Precision = 0.677

11 Point Average Precision = 0.681

Since the AP is very much less than 1.0, it would be good idea to look at the Scenes where the model failed to correctly predict. We discuss about such cases in Section 5.

4 Detections and Evaluation of RADAR dataset

4.1 BEV, Detector Model and Average Precision

The process workflow for RADAR data is similar to that followed in Section 3. We are provided with output weight vector for the RADAR set as .pth files which cab be plugged into the Darknet framework for predictions. Since we choose RADAR as the base co-ordinate system [MK19], we don't need to perform any transformation operations on the point cloud. Rest of the procedure remains the same for BEV transformation, detections by Darknet and computation of Average Precision. Fig 4.1 shows the output results of RADAR Detector in Scene 208. The Output image contains the plotted RADAR observation in BEV, Bounding Boxes for the Ground Truth Object (shown in green rectangles) and Bounding Box for detected Objects (shown in red colored rectangle).

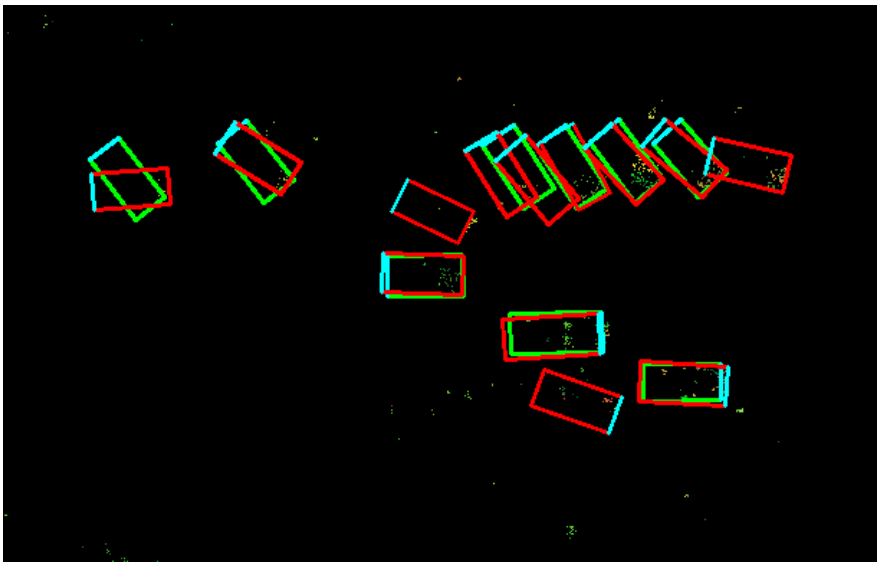


Fig 4.1 Detected and Ground Truth Objects in RADAR BEV (Scene 208) [MK19] [Git]

4.2 Result

Following are the results of RADAR validation data.

Average Precision = 0.668

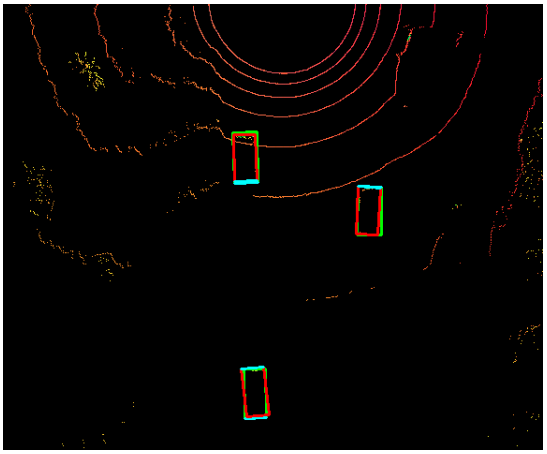
11 Point Average Precision = 0.658

Since the AP is very much less than 1.0, it would be good idea to look at the scenes where the model failed to correctly predict. We discuss about such cases in Section 5.

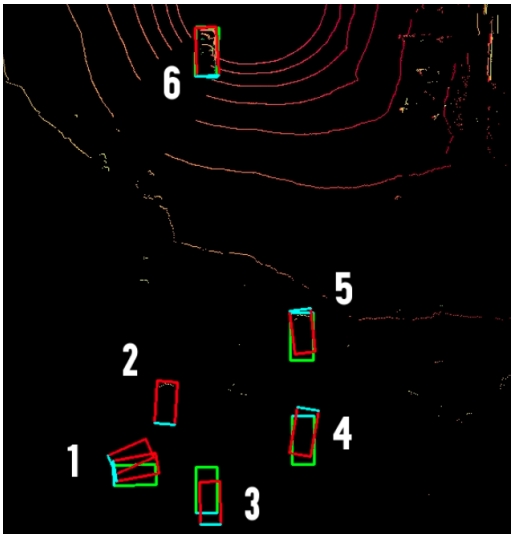
5 Detection Examples

5.1 LiDAR

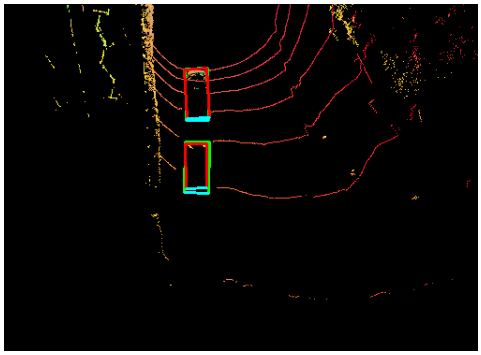
Following are four scenarios from the Object Detector's LiDAR output. In Fig 5.1A corresponding to Scene ID 110, the Ground Truth and the YOLO outputs match. There are 3 Objects in Ground Truth and YOLO also predicts 3, bounding boxes of which are very much overlapping with Ground Truth. In Fig 5.1C also, corresponding to Scene ID 259, the Ground Truth and the YOLO outputs match. There are 2 Objects in Ground Truth and YOLO also predicts 2, bounding boxes of which are very much overlapping with Ground Truth.



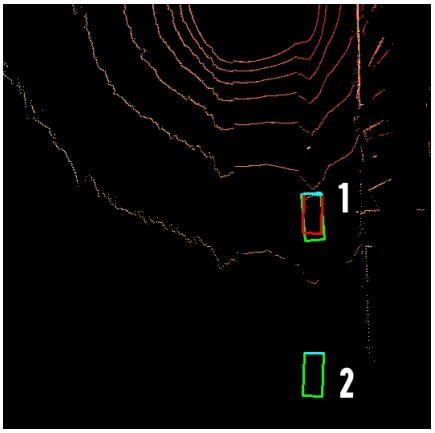
(a) Fig 5.1A Detections and Ground Truth (Scene 110)



(b) Fig 5.1B Detections and Ground Truth (Scene 242)



(a) Fig 5.1C Detections and Ground Truth (Scene 259)



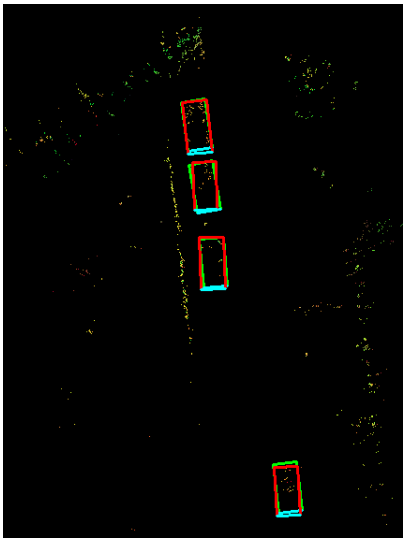
(b) Fig 5.1D Detections and Ground Truth (Scene 046)

On the other hand, in some cases like as shown in Fig 5.1B corresponding to Scene ID 242, the detections by YOLO are quite different from Ground Truth data. Here objects labelled as 2 and 6 are matching to Ground Truth, but object 3,4 and 5 has a bounding box which is translated from Ground truth. Object 1 is the worst where 2 bounding boxes are drawn by Detector, both of which are not overlapping with the Ground Truth. In Fig 5.1D corresponding to Scene ID 046, the prediction by the model have some flaws. Corresponding

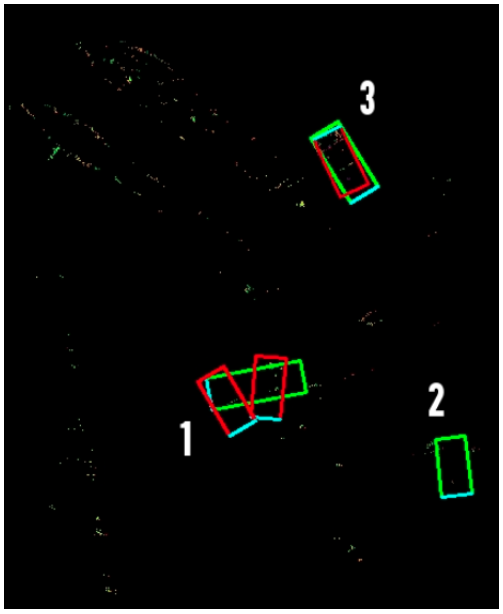
to the 2 Ground Truth Objects, the Detector predicts only 1 object correctly (marked as 1) whose Bounding Box are relatively matching. On the other side, it failed to predict 1 object (marked as 2) which can be termed as False Negative.

5.2 RADAR

Following are four scenarios from the Object Detector's RADAR output set. In Fig 5.2A corresponding to Scene ID 020, the Ground Truth and the YOLO outputs match. There are 4 Objects in Ground Truth and YOLO also predicts 4, bounding boxes of which are very much overlapping with Ground Truth. In Fig 5.2C also, corresponding to Scene ID 259, the Ground truth and the YOLO outputs match. There are 2 Objects in Ground Truth and YOLO also predicts 2, bounding boxes of which are very much overlapping with Ground Truth.

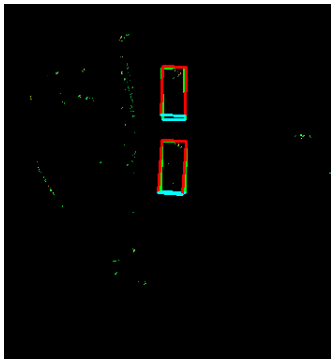


(a) Fig 5.2A Detections and Ground Truth (Scene 020)

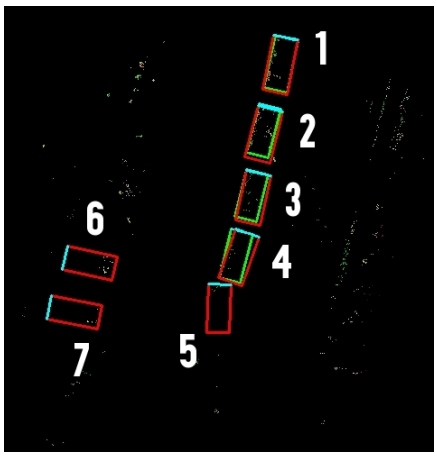


(b) Fig 5.2B Detections and Ground Truth (Scene 143)

On the other hand, in some cases like as shown in Fig 5.2B corresponding to Scene ID 143, the detections by YOLO are quite different from Ground Truth data. Here object labelled as 3 matches to Ground Truth, but object 1 is worse where 2 bounding boxes are drawn by Detector, both of which are not overlapping with the Ground Truth. There is not any prediction done by Detector at Ground Truth Object 2. In Fig 5.2D corresponding to Scene ID 031, the prediction by the model have some flaws. Although, corresponding to the 4 Ground Truth Objects, the Detector predicts 4 objects (marked as 1,2,3,4) whose Bounding Box are good match. But on the other side, it also predicted 3 extra objects (marked as 5,6,7) which can be termed as False Positives as no Ground Truth is present corresponding to these 3 predictions.



(a) Fig 5.2C Detections and Ground Truth (Scene 259)



(b) Fig 5.2D Detections and Ground Truth (Scene 031)

6 Future Scope

The results of the model in validation dataset are not the best. It might be a good idea in future to train the model using a different setup of Neural Network such as ResNET, which is more complex network. We could also use this setup using Transfer Learning to just retrain a few layers of a network on our source domain of Astyx dataset. As the reference domain we can use a network trained on larger dataset such as nuScene [Cae+20] or Waymo Open [Wayb] which has millions of Images.

Articles

[Die95] Tom Dietterich. "Overfitting and Undercomputing in Machine Learning". In: *ACM Comput. Surv.* (Sept. 1995), pp. 326–327.

[Rez+19] Hamid Rezatofighi et al. "Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression". In: (Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019).

[Zan+19] S. Zang et al. "The Impact of Adverse Weather Conditions on Autonomous Vehicles: How Rain, Snow, Fog, and Hail Affect the Performance of a Self-Driving Car". In: *IEEE Vehicular Technology Magazine* 14.2 (2019), pp. 103–111.

[BR20] F. Berens and Hochschule Ravensburg Weingarten. "Lecture notes in Lidar and Radar Systems, How to Astyx Complex Yolo". In: (Nov. 2020).

[Cae+20] Holger Caesar et al. "nuScenes: A Multimodal Dataset for Autonomous Driving". In: (Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) June 2020).

[ER20] S. Elser and Hochschule Ravensburg Weingarten. "Lecture notes in LiDAR and RADAR Systems, Chapter 1:Introduction to ADAS and ADS". In: (Oct. 2020).

[MK19] M. Meyer and G. Kuschk. "Automotive Radar Dataset for Deep Learning Based 3D Object Detection". In: (16th European Radar Conference (EuRAD) 2019), pp. 129–132.

[NV06] A. Neubeck and L. Van Gool. "Efficient Non-Maximum Suppression". In: 3 (18th International Conference on Pattern Recognition,2006), pp. 850–855.

[Red+16] J. Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: (IEEE Conference on Computer Vision and Pattern Recognition (CVPR),2016), pp. 779–788.

[ZZ09] Ethan Zhang and Yi Zhang. "Average Precision". In: (Springer US 2009). Ed. by LING LIU and M. TAMER ÖZSU, pp. 192–193.

Webpages

[Git] Github. *sa32953/lidar*. URL: `https://github.com/sa32953/lidar`. (accessed: 10.12.2020).

[Gmb] Astyx GmbH. *Astyx HiRes2019 Dataset*. URL: `https://www.astyx.com/development/astyx-hires2019-dataset.html`. (accessed: 10.12.2020).

[Ind] Jaccard Index. *Jaccard Index*. URL: `https://en.wikipedia.org/wiki/Jaccard_index`. (accessed: 21.01.2021).

[M D] et al M. Droettboom. *MatplotLib*. URL: `https://matplotlib.org/`. (accessed: 10.12.2020).

[Num] Numpy. *NumPy*. URL: `https://numpy.org/`. (accessed: 10.12.2020).

[OSP] OSPython3. *os*. URL: `https://docs.python.org/3/library/os.html`. (accessed: 20.01.2021).

[PyT] PyTorch. *PyTorch*. URL: `https://pytorch.org/`. (accessed: 20.01.2021).

[Sha] Shapely. *Shapely*. URL: https://pypi.org/project/Shapely/. (accessed: 20.01.2021).

[Waya] Waymo. *Waymo*. URL: https://waymo.com/. (accessed: 10.12.2020).

[Wayb] Open Dataset - Waymo. *Waymo Inc*. URL: https://waymo.com/open/. (accessed: 21.01.2021).

[Wika] Wikipedia. *Lidar*. URL: https://en.wikipedia.org/wiki/Lidar. (accessed: 10.12.2020).

[Wikb] Wikipedia. *RADAR*. URL: https://en.wikipedia.org/wiki/Radar. (accessed: 20.01.2021).

[Wikc] Bird's Eye View- Wikipedia. *Bird's Eye View- Wikipedia*. URL: https://en.wikipedia.org/wiki/Bird's-eye_view. (accessed: 21.01.2021).

[R16] Joseph R. *Darknet: Open Source Neural Networks in C*. 2013–2016. URL: http://pjreddie.com/darknet/. (accessed: 20.01.2021).