Pre-thesis -II Report

Occluded Object Detection for Autonomous Vehicles using the YOLO model

Name and ID:

1) Tanzim Mostafa, 18201151
2) Sartaj Jamal Chowdhury, 18201160

Supervisor: Dr. Md. Khalilur Rhaman

Co-supervisor-1: Dr. Md. Golam Rabiul Alam

Co-supervisor-2: Moin Mostakim

Date of Submission: 19/1/2022

Department of Computer Science and Engineering

BRAC University

Occluded Object Detection for Autonomous Vehicles using the YOLO model

Abstract

Autonomous vehicles [AVs] are the future of transportation and they are likely to bring countless benefits compared to human-operated driving. However, there are still a lot of advances yet to be made before these vehicles can be considered completely safe and before they can reach full autonomy. Perceiving the environment with utmost accuracy and speed is a crucial task for autonomous vehicles, ergo making this process more efficient and streamlined is of paramount importance. In order to perceive the environment, AVs need to classify and localize the different objects in the surrounding. For this research, we deal with the detection of occluded objects to help enhance the perception of AVs. We introduce a new dataset containing occluded instances of road scenes from the perspective of Bangladesh and evaluate the YOLO model, more specifically YOLOv5, with this dataset. The results demonstrate the effectiveness of our model used as well as the dataset.

I. Introduction

Although autonomous vehicles have become popular to most of us in recent years, the first evidence of driverless vehicles dates back to 1925, when the “American Wonder" was demonstrated by Houdina Radio Control. This was a remote-controlled car being operated by a person in a vehicle just behind it, as it was driven along Broadway in New York City [1]. Then in 1995, a major milestone was reached by Carnegie Mellon University's Navlab team when they drove from Washington, D.C., to San Diego, CA, with 98% automation using manual longitudinal control [1]. From the 1990s to the early 2000s, there were several other advancements in this field.

In 2012, for the first time, researchers around the world could evaluate their progress on the many different self-driving perception problems in an effective way since the KITTI Vision Benchmark was made accessible to the public [2]. It was also at this time that deep learning started to make a profound impact on several fields such as computer vision and robotics. This created a fundamental base for notable breakthroughs in the perception elements of AVs, in terms of robustness, accuracy, run-time, etc.

In 2014, a classification of autonomous driving systems was introduced by the Society of Automotive Engineers (SAE). It comprises six levels of SAE autonomy, where level 0 means no autonomy and level 05 means full autonomy.

From then on, self-driving cars by Mercedes, Tesla, Google's Waymo, among others, were released.

Autonomous vehicles or self-driving cars have the potential to bring about a revolutionary impact on society and the way people travel. They provide many benefits such as more safety, reduced carbon emission, reduced traffic congestion, greater independence such as for people with disabilities, the elderly, and so on [3].

In recent years, we have seen numerous advances in autonomous vehicles. For example, we all know about the autopilot feature in Tesla cars through which it is able to steer, accelerate and brake automatically in its lane. Although, the car is not fully autonomous and requires some driver supervision. Similarly, there exist several other companies that are in the process of building their own self-driving cars. It is therefore without a doubt that fully self-driven vehicles are on the horizon.

Autonomous vehicles make use of certain sensors such as radar, lidar, and cameras in order to perceive the environment around them. Radars are cheap, ultrasonic, and work effectively in extreme weather conditions. Lidar is the most powerful sensor and provides very accurate depth data, but it is very expensive [4]. Cameras work well in clear, good lighting conditions but they have substandard performance in darkness and extreme weather and are bad at depth estimation.

There exist two different philosophies among Tesla and Waymo, two companies that are at the forefront in the development of self-driving technology, regarding their main perception sensor. While Waymo's driving system is more dependent on Lidar sensors, Tesla argues that cameras are sufficient and don't use Lidars in their cars [5].

There are still several challenges that have to be tackled before self-driving cars can reach full autonomy. Among them, being able to accurately perceive the environment around them and detect occluded objects, is a significant one.

A) Research Problem

Deep learning is a subset of Machine Learning (ML) and Artificial Intelligence (AI). It is a multi-layered computational model that is used to automatically extract features and patterns from raw data, such as images, and predict, or take actions based on a certain reward function. Techniques such as neural networks, hierarchical probabilistic methods, supervised and unsupervised learning models, and deep reinforcement learning (DRL) are all subtypes of DL. DL has recently become extremely popular because of the remarkable results achieved in the fields of object detection, natural language processing, image classification, etc. There are mainly two causes behind the current rise of DL: the availability of large datasets and high-performance Graphics Processing Units (GPU) [6].

Autonomous vehicles have to detect various objects in the road in order to achieve a complete view of the environment. Deep learning-based perception, mainly Convolutional Neural Networks (CNNs), is the existing standard for object detection and recognition. Various types of neural network architecture are used for object detection as segmented areas in images based on pixels or 2D regions of interest, 3D bounding boxes in lidar point clouds, and 3D representations of objects using both camera and lidar data [5].

Image data is more suitable for object recognition as it contains richer information, and this is what will be used primarily in our research. Although, in this case, the 3D positions of the detected objects in the real world have to be estimated because the depth information is lost when the imaged scene is projected onto the imaging sensor. In the case of 2D object detection, single-stage and double-stage detectors are the most popular architectures [5].

According to [4], some of the perception intricacies that require further research are:

- Partial view and angled view
- Occluded object
- Many objects in the surrounding
- Similar objects at different distances
- Various lighting conditions depending on the time of day
- Road conditions such as unclear road markings and slippery road
- Adverse weather conditions such as rain, fog, snow, etc
- Changing traffic lights
- Lane markings that have faded
- Requirement for invariances such as rotational invariance, translation-invariance, size, and color invariance

To briefly explain some of these complexities and more:

When similar objects are at different distances, it seems like they are of distinct sizes, when actually they belong to the same class category [7].

Adverse weather conditions such as rain, fog, or snow, majorly degrade the performance of object detection methods. This is because the quality of visual signals that are sensed by self-driving cars is reduced in such conditions. Solving the effect of rain in the context of autonomous driving, especially, is an area that is popular among various research communities [8].

Transformations such as translation, rotation, scale, etc, of an object in an image, make it difficult to recognize or detect an object. Such a transformation problem is also known as the

“invariance problem” and there is still much work to be done before we are able to find a strong and scalable solution to this problem using machine learning [9].

Illumination has a significant impact on detecting objects. For example, cameras are unable to capture good-quality images in low light conditions. Most of the current object detection methods do not have high accuracy in poor lighting conditions [10].

Detection of small objects is challenging because it is difficult to differentiate the objects from the background and similar objects. Also, the small object can be placed at a great number of possible locations on a given image, so accurately locating the objects requires more powerful computation. Moreover, knowledge on small object detection is very limited because most previous efforts were on large object detection problems [13].

Occlusion can be present in various ways and can range from partial occlusion to heavy occlusion. In the case of an autonomous driving environment, target objects can be occluded by static objects such as lampposts. Moving objects such as cars may occlude each other, or even self-occlude, such as when parts of a cyclist overlap [11]. Pedestrian detection is a part of occluded object detection and is still a complex problem because of the diversity of occlusion patterns [12].

While many of the object detection architectures have already achieved amazing accuracy and extremely low error rates of less than 5%, on datasets such as Pascal VOC and ImageNet, another problem is the speed of these architectures in producing low error rates in real-time in the case of autonomous driving [4].

So clearly, there is room for improvement in a lot of areas regarding object detection in autonomous vehicles. In this paper, we are primarily going to focus on overcoming the problem of occluded object detection while also trying to maintain a high detection speed. We aim to do this by creating our own dataset of occluded objects from road scenes in Dhaka and using the most recent YOLO model, the YOLOv5, to evaluate its performance on our dataset.

B) Research Objectives

This research aims to develop a system that can increase the accuracy of occluded object detection, while also maintaining high detection speed. The objectives of this research are:

1. To deeply understand the perception problem in autonomous vehicles.
2. To deeply understand how deep learning can be used for object detection in autonomous vehicles.

3. To develop a dataset that can be leveraged to evaluate, and help improve the performance of miscellaneous deep learning models for occluded object detection.
4. To specifically evaluate the YOLOv5 model on our dataset.
5. To offer suggestions on how to improve the overall performance

II. Literature Review

Autonomous vehicles have numerous benefits and are likely to bring a far-reaching impact on the way people travel. Companies such as Tesla, Waymo, Uber, nuTonomy, etc, predict a future with self-driving cars by the next 15-20 years [4].

However, there still exist significant challenges before we can completely rely on self-driving cars and before they can reach full autonomy. One of these challenges is being able to perceive the environment with high accuracy even in complex situations. Object detection falls under perception and autonomous driving requires very high detection accuracy as well as speed in the given context [14].

2.1. Deep Learning for Object Detection

Most of the recent advances in object detection are accredited to advances in deep learning. More specifically, CNN-based object detection algorithms are most popular for detecting objects, rather than classical detection algorithms [12]. While these algorithms already work well in many cases, when it comes to occluded object detection, most of these models have been unable to produce very good results.

2.1.1 Deep Learning based Object Detection Architectures

Single-stage and double-stage detectors are the architectures used most often for 2D object detection [5]. Single-stage detectors only need a single pass through the neural network and predict all the bounding boxes in one go. For example, You Only Look Once (YOLO), Single Shot MultiBox Detector (SSD), RefineNet, etc. Double stage detectors divide the process of object detection into two sections: region of interest candidate proposals and bounding boxes classification. Examples include RCNN, Faster-RCNN, R-FCN, etc.

2.2. Related Works

This part aims to critically review previous relevant work regarding occluded object detection in autonomous vehicles. We analyze the different techniques used to achieve better results and show that there are still a lot of challenges that need to be met regarding accuracy, speed, etc.

In the paper [14], the authors used Multi-Scale CNN(MS-CNN) as a baseline network and added 3 enhancements to overcome the occluded object detection and large object scale variation

problem. Their network follows 2 stage object detection architecture. According to the paper, the classical non-maximal suppression (NMS) method, which is used to select a single bounding box out of overlapping proposals, has shortcomings when it comes to detecting occluded objects. Hence, they used another method called soft-NMS, at object proposals from various feature output scales so that they can achieve a balance on the quality and number of object proposals. They evaluated their system over the KITTI 2D object detection dataset and were able to achieve better object detection performance with almost no increase in detection time. For example, it was first for the pedestrian detection category “Easy” and second place for “Moderate” and “Hard”.

The research work [12], proposes some improvements in the feature extraction subnet and detection subnet, in both two-stage and one-stage architectures, which can lead to better object detection. According to the authors, in terms of improving the feature extraction subnet, adding more features such as detailed and context features, can help detect small objects and occluded objects with higher precision. As an example they reviewed HyperNet to add more detailed features and Inside-Outside Net(IONNET) or Multi-Scale Deep Convolution Neural Networks(MS-RCNN), to add context features. In terms of improving the detection subnet, they propose adding output branches other than classification and localization and using Soft-NMS or IoU-Net instead of the classical NMS. To talk more specifically about occluded object detection, they also suggest that data augmentation can be used to increase the number of occlusion examples and make the system more robust to occluded object detection. Other than that, adding context features, relation features, and improving the loss function, such as using Repulsion Loss, can all help in achieving higher accuracy in detecting occluded objects.

The paper [15] uses an object detection network consisting of two modules. One of which is an object detection framework that performs classification and bounding box regression. The other one uses multiple OBB (Object Bounding Box) Critic networks which divide object areas and occlusion areas. Furthermore, they used the Faster R-CNN Object Detection framework as a base method and evaluated it on the KITTI Vision Benchmark Suite Dataset. The proposed method detected hidden or occluded objects with greater accuracy compared to the Faster R-CNN methods. However, the paper did not focus on comparing the computation time and memory consumption of the different object detection methods used, even though the running time and memory complexity are important factors for object detection in road scenes in real-time.

The research [18] designed a good dataset namely Dhaka AI, which covers a vast amount of Dhaka traffic data. However, a dataset heavy with occlusion instances from the road scenes of Dhaka, and specifically made to simulate the perception of autonomous vehicles is unattainable.

From the above discussion, it is observed that although some of the models proposed were able to achieve better detection accuracy for occluded objects, there is still space for much

improvement. Another particular challenge that arises in proposing some of the enhancements to address the occluded object detection problem is that the network becomes too complex and so the detection time increases due to more expensive computation requirements. Therefore, in this research, we proceed to propose a novel dataset that can be leveraged in the evaluation of different deep learning models and measure their performance on the aforementioned problem. We then use the latest version of the YOLO model, the YOLOv5, in order to evaluate its performance on this dataset.

III. Methodology

In order to effectively describe our research methods, we have divided this section into three parts, namely, workflow, dataset, and model.

3.1. Workflow

Our research involves producing an adequate dataset that would aid autonomous vehicles in occluded object detection. Figure 3.1 portrays the overall workflow of our methods. First, we obtain data, then handpick images of occluded instances, annotate those images, split the dataset into train and test sets, preprocess and augment them, as shown, which will be further elaborated in the dataset subsection. After that, we train our model of choice and evaluate it based on the test sample. Finally, based on our results we plan to bring further enhancements to the dataset.

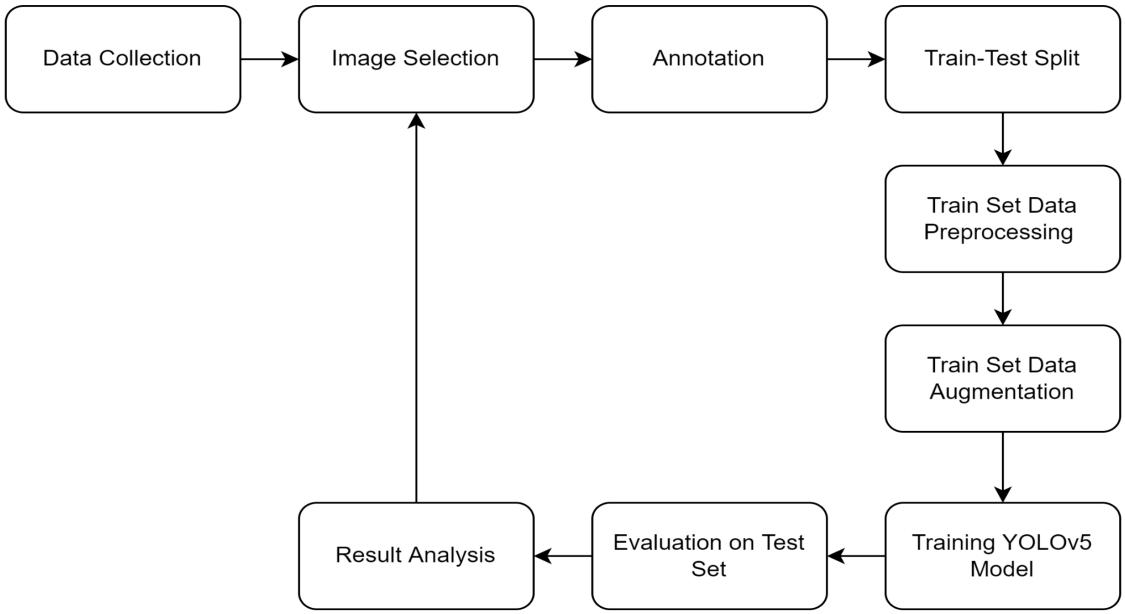

Figure 3.1: The overall workflow

3.2. Dataset

To design our dataset we had to carry out some groundwork for data collection. Also, we had to come up with a number of decisions on how to process the data, acquire the most relevant images, estimate a sensible volume of data to be annotated, etc. After manually annotating the images we had to determine how to split, preprocess, and augment the data. In what follows, we describe the various steps and decisions made on designing the dataset.

3.2.1. Data Collection

At the same time-stamp, we recorded road scenes from the busy roads of Dhaka from 3 different vehicles: an SUV, a wagon, and a minivan. We recorded the road scenes for 1 hour and 20 minutes, while each of the cars maintained different lanes and remained abreast as much as possible. Also, we did maintain a unique camera angle from each vehicle, but we used the same type of camera (iPhone 11's back-facing camera in HD 30 fps mode) on each of them. Therefore, we got data from 3 different perspectives, resulting in a total of 4 hours of video data. Our goal was to simulate the perception of autonomous vehicles. Subsequently, we recorded scenes from 3 distinctive angles of the forward direction, as shown in Figure 3.2. This helped us have a more diverse dataset with miscellaneous occlusion instances of the same objects from various directions.

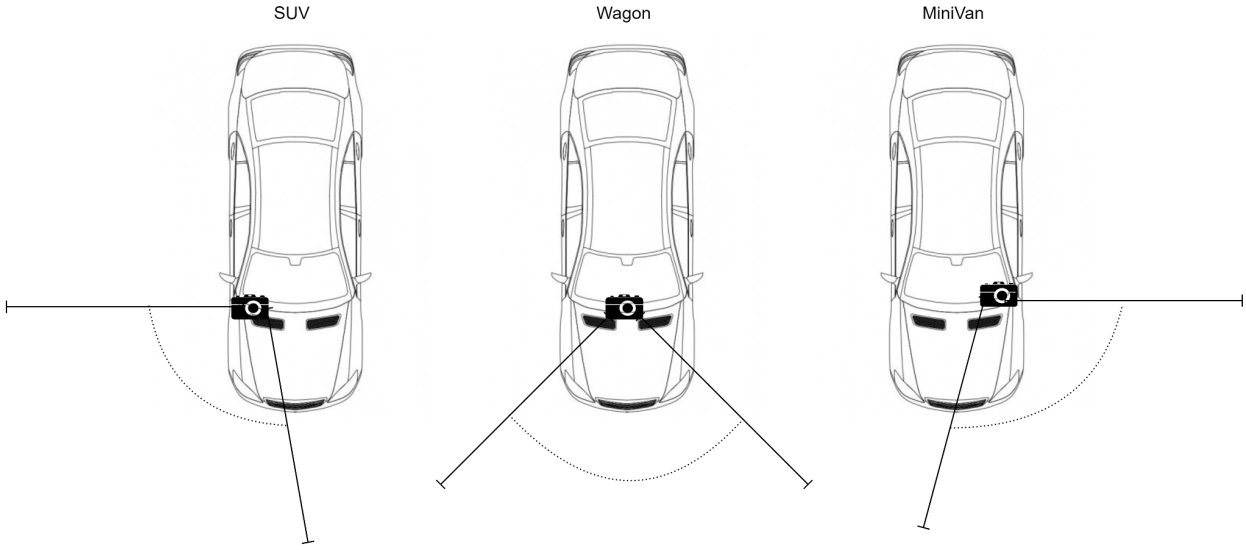

Figure 3.2: Three unique camera angles for each car used to collect data

3.2.2 Image Selection

We converted the videos to images on 10 frames per second basis. After that, we manually selected images consisting mostly of occluded instances of every class of objects. Most images

for example had relatively faraway objects (cars, rickshaws, etc) blocked by more nearby objects. Moreover, we made sure that a multitude of occlusion levels was present within the dataset.

3.2.3 Annotation

We used 2D bounding boxes for annotation because our goal was to detect multiple occluded objects from all the images. We aim to annotate a total of 1500 images. However, we have a total of 667 annotated images so far. We used the online tool Roboflow Annotate because it is efficient and provides miscellaneous image dataset preparation functions. After generating the dataset, an API call was made from Google Colaboratory to train the dataset using a YOLO model.

3.2.4 Labeled Classes

We annotated 8 different object classes, by maintaining a balance between the typical ones. We made sure to cover a large part of each image so that there were not many void classes, while also trying to reduce our endeavors to annotate due to time constraints. However, we did carefully annotate every instance with occlusion. Table 3.3 shows the list of different classes of objects we labeled:

Class	Description	# of Labels
person	Both pedestrians and visible riders of certain vehicles.	1043
car	All four-wheelers except buses.	604
rickshaw	Only rickshaws.	500
other-vehicle	Any three-wheeler or two-wheeler except CNG, rickshaw, motorcycle, and bicycle.	172
cng	A common three-wheeled vehicle in Dhaka that is similar to an auto-rickshaw.	167
motorcycle	All types of motorcycles. The rider is not included in the bounding boxes but rather labeled as a “person”.	138

bus	Only buses.	129
bicycle	All types of bicycles. The rider is not included in the bounding boxes but rather labeled as a “person”.	85

Table 3.3: List of different classes of objects we labeled

3.2.5 Train-Test Split

We decided to split the data into 85% training and 15% validation sets. The terms validation and test split are often interchangeable, both of which refer to the segment of the dataset that was not used to train the model or tune the model parameters. Therefore, it provides an unbiased evaluation of the model. In our experiment, we used the training sample for the model training and the validation sample as a test set to estimate the performance quality.

3.2.6 Preprocessing and Training Data Augmentations

In order to reduce the training time of the model, we resize the images from 1920*1080 to 768*432. Generally, machine learning models can train faster on smaller images. However, we had to make sure the images were not too small and through experimentation, we concluded that a 60% reduction of both height and width, keeping the aspect ratio invariant, would roughly strike a good balance between the training speed and detection accuracy.

We augmented the training data so that we could feed our model with a further variety of instances. First, by incorporating a horizontal flip to all the images. Then, adding grayscale versions of 50% of them because grayscale images can result in higher accuracy classification across a multitude of different classifiers according to [16]. Finally, we leveraged images with blur up to 1.5 pixels since [17] helps justify that manipulating color, contrast, and blur can help the model gain abilities to generalize noise patterns in the train set samples that were not seen before.

Our dataset can be downloaded from the following link:
https://app.roboflow.com/ds/RV7qjxi6nn?key=kZj9n6BYu9

3.3 Model

As mentioned before, there exist several state-of-the-art deep learning-based object detection models. These models can mainly be divided into two types: two-stage detectors and one-stage

detectors. Two-stage detectors include Faster R-CNN, Mask R-CNN, etc, while one-stage detectors include YOLO, SSD, and so on.

The main difference between two-stage and one-stage detectors is in accuracy and speed. In two-stage detectors, the entire training occurs in two steps. In the first step, various regions of objects are extracted and in the second step the objects are classified and the localization of the objects is made more accurate. Because of the two stages, such detectors are often more accurate but are computationally more expensive. On the other hand, in one-stage detectors, there is no separate stage for the extraction of different regions. Instead, a fixed number of predictions are made on the grid and the whole detection happens in one go. This makes one-stage detectors generally faster than two-stage detectors, although with a slight decrease in accuracy. Hence, they are also more suited for real-time detection tasks.

For our object detection task, we decided to use the YOLO model, more specifically YOLOv5, which is a single-stage detector. This is because our object detection is for the use of autonomous vehicles where fast detection speed is crucial. While there exist other single-stage detectors like SSD, EfficientDet, we decided to use this, as it is one of the fastest models that exist and also very accurate. According to [19], YOLOv4 has been deemed the best real-time object detection model in 2021 judging by the Mean Average Precision (mAP) metric on the MS COCO dataset. In the paper [20], it says that YOLOv4 runs two times faster than EfficientDet but with similar performance. YOLOv5 is the most recent version by Glenn Jocher but has been released without a paper alongside. Glenn Jocher first implemented a version of YOLOv3 in Pytorch and had been constantly making improvements to the architecture. Eventually, YOLOv5 was officially released by a company called Ultralytics on 25th June 2020, which has been shown to be better in terms of speed and precision than YOLOv4 in the COCO benchmark [21].

Since no paper has been released about YOLOv5, the exact architecture used is still not completely known. Hence, in the following, we give a basic idea about how the YOLO model works. The YOLO model treats the object detection task as a single regression model. It is based on a backbone model whose role is to extract meaningful features from the image that will eventually be used by the final layers, as we can see in figure 3.4. For example, the original YOLO model used a modified version of GoogleNet as the backbone, which was later changed to DarkNet networks in the later models by Joseph Redmon.

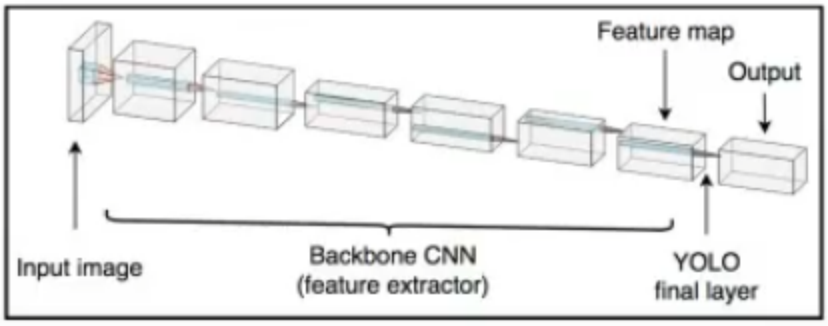

Figure 3.4: Basic structure of the YOLO model

First, the input image is divided into an S*S grid. Every grid cell is then responsible for detecting objects that appear in them. For instance, if the center of an object appears within a grid cell, then it is the job of that cell to detect it.

Each grid cell gives a number of bounding boxes, which is a rectangular outline that is used to focus on the object present in an image. Then, the job of each cell is to predict the following for each bounding box: the center of the box, the height and width of the box, the probability that an object exists in this box, and the class of that object. By class, we mean that it could be a cat, dog, person, etc. It uses a single bounding box regression in order to predict the center, width, height, and class.

After that, it uses the concept of Intersection over union (IOU) to remove bounding boxes lower than a specific threshold. IOU in object detection determines how the bounding boxes overlap. If the bounding box given as output is equal to the ground truth bounding box, IOU is 1 or maximum. The less the output box and ground-truth box match, the lower the IOU value. This concept is used to get rid of bounding boxes that are further apart from the real box.

The algorithm also uses a technique called Non-Max Suppression (NMS). There may still be multiple detections made for the same object. NMS technique makes sure that only one bounding box is provided for each object.

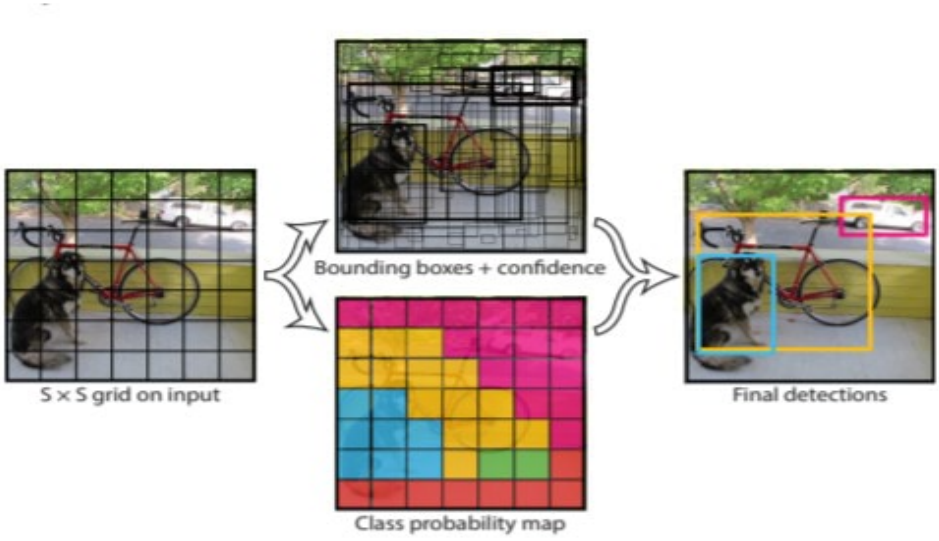

Figure 3.5: Basic structure of the YOLO model

Figure 3.5 taken from [22] depicts how everything works. As we can see, in the end, we are left with perfectly fitted bounding boxes for the dog, bicycle, and car classes. It also outputs the probability with which each class is detected.

A significant part of the process is the loss function. During training, the YOLO algorithm optimizes the following loss function:

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbf{1}_{i}^{\text{obj}} \sum_{c \in \text{ classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
$$

To explain a little about the loss function, we divide it into three parts. The first part of the equation:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

represents the bounding box loss. It basically aids the network to learn the weights which are used to predict the bounding box coordinates and size.

The next part:

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

represents the object confidence loss. This is responsible for teaching the network to learn the weights so that it can predict whether or not a bounding box contains an object.

The last part of the equation:

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{ classes}} (p_i(c) - \hat{p}_i(c))^2$$

represents the classification loss. This makes sure that the network learns to predict the correct class for each of the bounding boxes.

This is just a brief description of how the YOLO model works. The newer versions of the model introduced several new features that have helped the models become faster and more accurate.

IV. Implementation and Results

This section describes the implementation of the YOLOv5 model for occluded object detection using our dataset, as well as our results. The entire coding was carried out in a Google Colaboratory notebook. We mainly ran python scripts and used PyTorch, which is an open-source machine learning library used for Computer Vision tasks, to implement our model and perform object detection. The code can largely be divided into two stages. In the first stage, we import the YOLOv5 repository and our dataset. After that, we train our model on our training set and test the model on our validation set.

4.1. Implementation

First, we import some of the libraries that will be required for our task such as “torch” for using PyTorch, “os” for managing directories, and “IPython.display” so that we can display our detected images. We keep an option for using GPU if it is available or else use CPU. Using the GPU provided by Google Colab significantly reduced the training time of the model. We also install some other dependencies to be able to use the YOLOv5 model and Roboflow, which is where we created our dataset.

4.1.1. Downloading YOLOv5 repository and Dataset in Google Colab

After cloning the repository of YOLOv5 by Ultralytics, a folder is created in our Colab notebook named “yolov5”. Inside this folder, there are various subfolders and python files that we will use later on. One of the subfolders is named “models'', where further variants of YOLOv5 are present like yolov5s, yolov5m, yolov5l, etc. These variants refer to some differences in terms of the complexity of the architecture of yolov5. For our task, we will be using the yolov5s variant, where the s stands for small, as this will be capable enough to do our specific task.

In order to download our dataset from Roboflow, we have to make an API call to Roboflow. Roboflow provides us with the code snippet required to make this call and download our dataset when we choose to export our dataset from Roboflow. After running the series of codes, our dataset folder is created in our notebook, containing train and validation subfolders. Each of these subfolders contains the “.jpg” files of images along with their “.txt” files for labels.

Object Class	x_center	y_center	bbox_width	bbox_height
1	0.03125	0.7280092592592593	0.057942708333333336	0.09953703703703703
1	0.09049479166666667	0.7349537037037037	0.06380208333333333	0.11342592592592593
7	0.020182291666666668	0.7569444444444444	0.040364583333333336	0.11574074074074074
3	0.0546875	0.7569444444444444	0.048177083333333336	0.08449074074074074
2	0.15364583333333334	0.8078703703703703	0.111328125	0.17824074074074073
3	0.5455729166666666	0.8344907407407407	0.06705729166666667	0.1423611111111111
2	0.423828125	0.8344907407407407	0.248046875	0.3275462962962963
6	0.7604166666666666	0.8692129629629629	0.032552083333333336	0.15046296296296297

Table 4.1: An example of a “.txt” label file

Table 4.1 shows how one ".txt" label file looks like corresponding to an image. The eight rows mean that there are eight objects present in this specific image. The first column of each row represents the object class, the next two columns represent the x and y coordinate of the center of the box and the last two columns represent the width and height of the bounding box respectively. The YOLO model basically uses these label files while training the model and also to see how it performs during testing.

4.1.2. Training and testing the yolov5s model

In order to train our yolo5s model we run the following python script:

```
!python train.py --img 768 --batch 16 --epochs 100 --data
{dataset.location}/data.yaml --weights yolov5s.pt --cache
```

Here we mention that we are using images of size 768 pixels wide. We use a batch size of 16 and train our model for 100 epochs. We also mention the dataset location and that we are using the yolov5s variant. It took us about 24 minutes to train our model on 628 images of the training set.

After that we test our model using the following piece of code:

```
!python detect.py --weights /content/yolov5/runs/train/exp5/weights/best.pt
--img 768 --conf 0.5 --source {dataset.location}/valid/images
```

Here we mention the location of the best version of our updated weights after training. We again mention the image size of 768 pixels wide. We use a confidence score of 0.5, meaning that if our model makes a prediction with more than 0.5 probability about the object, then it will be considered correct and displayed. We also have to mention the directory of our validation images in the detection script.

After testing our model we can finally display the inference images of our validation set.

4.2. Results

We got the performance of our model by using TensorBoard, which provides visualizations for different metrics of machine learning tasks.

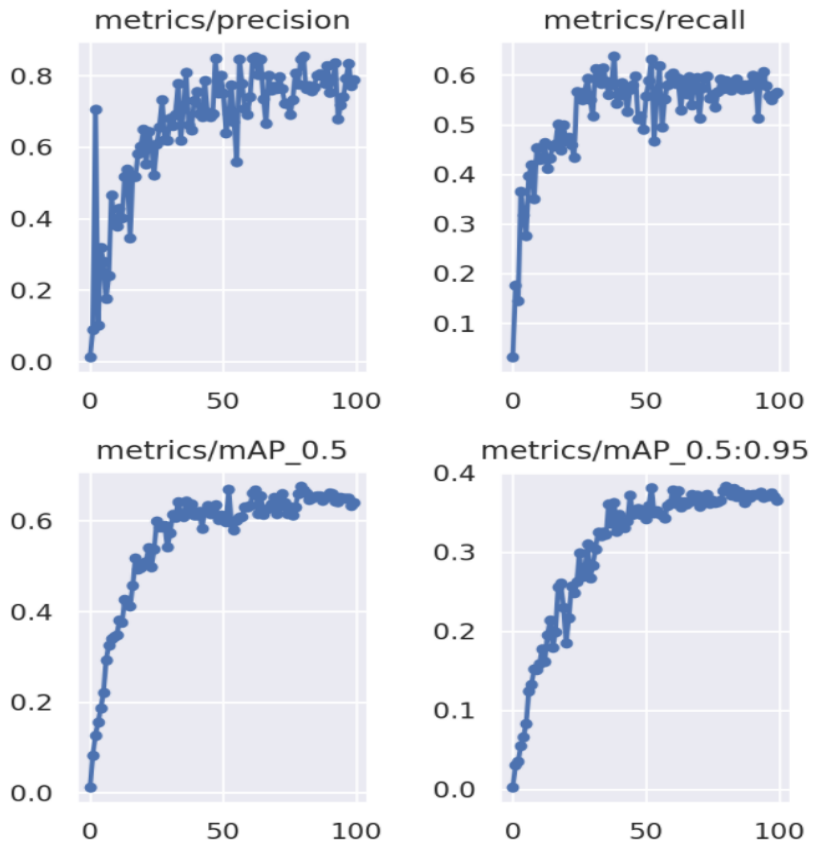

Figure 4.2: Plots of Precision, Recall and mAP metrics over the training epochs on our dataset

Mean Average Precision (mAP) is a common and popular metric used to measure the accuracy of object detectors. As we can see from figure 4.2, our model achieved a mAP_0.5 of about 0.65 and mAP_0.5:0.95 of about 0.36, at the end of 100 epochs. This means our model is good but does not have the highest accuracy. We also observe that our mAP_0.5 increased to about 0.65 after 35 epochs but did not increase further. We believe this is due to the fact that our dataset is not very large, especially for classes like bus, motorcycle, cng, bicycle, and other-vehicle, we did not have many instances in our dataset. We can also observe that our precision metric is about 0.78 and recall is 0.56 after 100 epochs. These are not substandard scores but they can definitely be improved further.

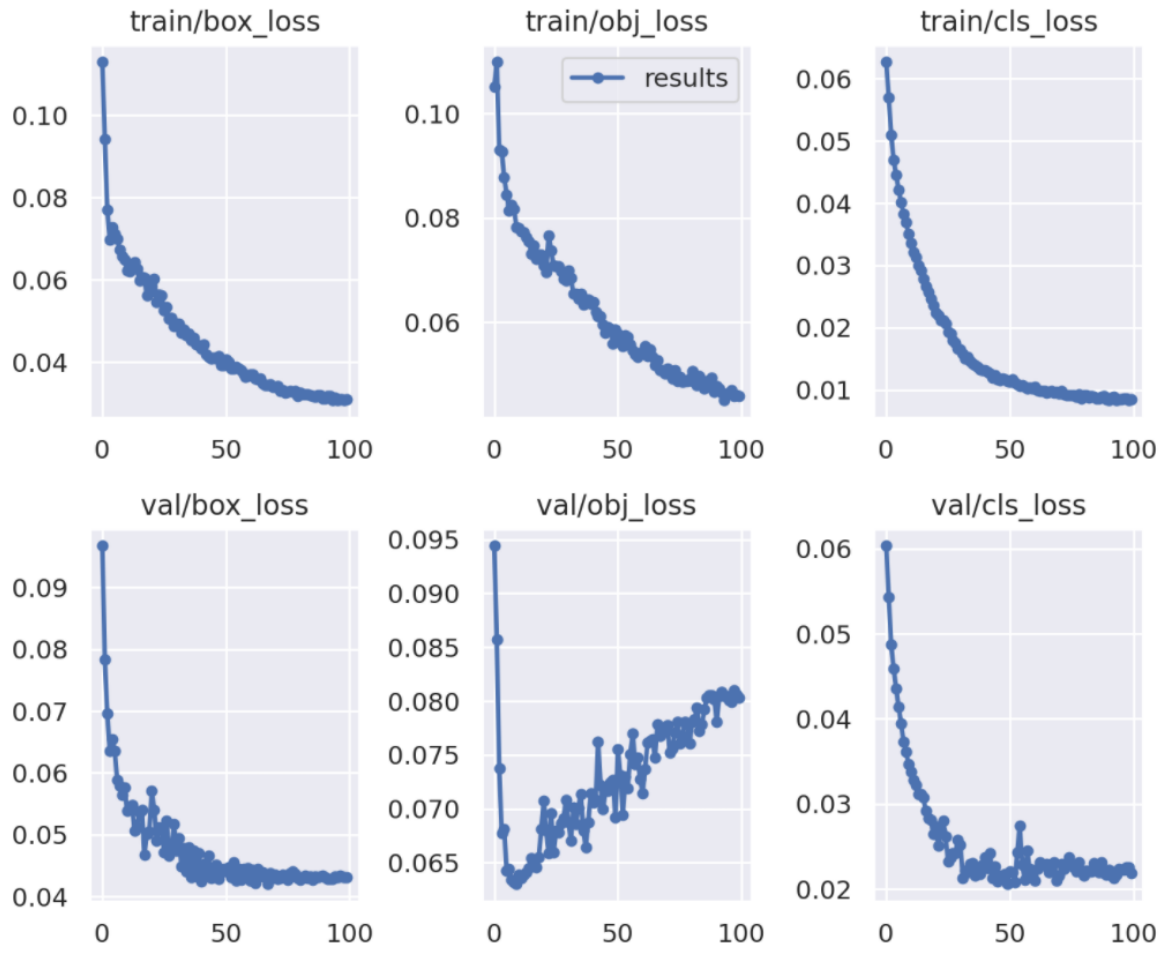

Figure 4.3: Plots of box loss, objectness loss, and classification loss over the training epochs for the training and validation set

Figure 4.3 shows three types of loss graphs, namely, box loss, objectness loss and classification loss. The box loss refers to how accurately our model was able to locate the center of the objects and how well the bounding box covered the objects. Objectness basically measures the probability that an object is present in a region proposed by the algorithm. Finally, classification loss measures how effectively the algorithm predicted the correct class of any object. The box, objectness and classification loss of the training set decreased rapidly until about 25 epochs and by the end of 100 epochs it was very low. The box and classification loss of the validation set also decreased rapidly and was very low by the end of 100 epochs. Our objectness loss of the validation set, on the other hand, started increasing after about 10 epochs and was about 0.080 by the end of 100 epochs.

Regarding the speed of our detections, we found that it took 0.5ms to pre-process and 14.0ms for inference. We could also see that it took a minimum of 0.012s and a maximum of 0.029s for

complete detections of each image in our validation set. Real-time detection means that the algorithm must detect at a speed of at least 30 frames per second or 0.033 seconds per image. Therefore, our model is having a very good detection speed and is suitable for autonomous vehicles.

If we look at some of the detected images from our validation set:

Figure 4.4: An example of occluded objects detected from a validation set image

From figure 4.4, we can see that our model has detected the right car occluded by the trees very well, with 0.94 probability.

Figure 4.5: An example of occluded objects detected from a validation set image

Also if we observe in figure 4.5, it has detected the rickshaw, cng, motorcycle, car, bus, and person classes effectively, even though some of them were quite occluded.

Figure 4.6: An example of occluded objects detected from a validation set image

As we see in figure 4.6, the car to the right that is occluded by the railing has been detected accurately but the bus was not detected.

After careful observation of all images, the main problem we find is when it comes to classes we do not have a high number of instances in our training dataset, our model is unable to detect occluded instances of them in all cases. We also believe this is the reason for not getting higher mAP scores.

V. Conclusion

We firmly believe that our dataset has managed to fulfill its purpose in terms of having an occluded object dataset for Bangladesh, so that it can learn to detect occluded objects such as rickshaws, cng, etc. Moreover, from the point of view of the crowded roads of Dhaka, being able to detect occluded objects can be a huge leap forward towards improving AV perception overall. Our model has been able to detect all the classes in general and also detected several classes of objects containing a high degree of occlusion very accurately. However, it has mainly found some difficulty in detecting occluded objects of classes like other-vehicle, cng, bicycle, bus, and motorcycle, where we did not have a large number of instances in our training dataset. We believe that increasing the size of our dataset, mainly the classes that were under-represented, can help improve the performance of our model significantly. This is what we aim to do in the future, as we already have sufficient recorded data but due to time constraints were unable to annotate them all. Finally, we also aim to use a federated approach in the future, where various vehicles and roadside units communicate with each other to share information. Later, using this data computations get executed, and model trained. We strongly believe that this approach also has a high potential to further improve the performance of occluded object detection for autonomous vehicles.

References

[1] Janai, J., Güney, F., Behl, A., & Geiger, A. (2020). Computer vision for autonomous vehicles: Problems, datasets and state of the art. Foundations and Trends® in Computer Graphics and Vision, 12(1–3), 1-308.

[2] Geiger, A., Lenz, P., & Urtasun, R. (2012, June). Are we ready for autonomous driving? the kitti vision benchmark suite. In 2012 IEEE conference on computer vision and pattern recognition (pp. 3354-3361). IEEE.

[3] Pettigrew, S., Fritschi, L., & Norman, R. (2018). The potential implications of autonomous vehicles in and around the workplace. International journal of environmental research and public health, 15(9), 1876.

[4] Gupta, A., Anpalagan, A., Guan, L., & Khwaja, A. S. (2021). Deep Learning for Object Detection and Scene Perception in Self-Driving Cars: Survey, Challenges, and Open Issues. Array, 100057.

[5] Grigorescu, S., Trasnea, B., Cocias, T., & Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. Journal of Field Robotics, 37(3), 362-386.

[6] Pathak, A. R., Pandey, M., & Rautaray, S. (2018). Application of deep learning for object detection. Procedia computer science, 132, 1706-1717.

[7] Uçar, A., Demir, Y., & Güzeliş, C. (2017). Object recognition and detection with deep learning for autonomous driving applications. Simulation, 93(9), 759-769.

[8] Hnewa, M., & Radha, H. (2020). Object Detection Under Rainy Conditions for Autonomous Vehicles: A Review of State-of-the-Art and Emerging Techniques. IEEE Signal Processing Magazine, 38(1), 53-67.

[9] Barrow, E. J. (2017). The Use Of Deep Learning To Solve Invariance Issues In Object Recognition (Doctoral dissertation, Coventry University).

[10] Qu, Y., Ou, Y., & Xiong, R. (2019, December). Low Illumination Enhancement For Object Detection In Self-Driving. In 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO) (pp. 1738-1743). IEEE.

[11] Gilroy, S., Jones, E., & Glavin, M. (2019). Overcoming occlusion in the automotive environment—A review. IEEE Transactions on Intelligent Transportation Systems, 22(1), 23-35.

[12] Lu, J., Tang, S., Wang, J., Zhu, H., & Wang, Y. (2019, June). A review on object detection based on deep convolutional neural networks for autonomous driving. In 2019 Chinese Control and Decision Conference (CCDC) (pp. 5301-5308). IEEE.

[13] Tong, K., Wu, Y., & Zhou, F. (2020). Recent advances in small object detection based on deep learning: A review. Image and Vision Computing, 97, 103910.

[14] Wei, J., He, J., Zhou, Y., Chen, K., Tang, Z., & Xiong, Z. (2019). Enhanced object detection with deep convolutional neural networks for advanced driving assistance. IEEE transactions on intelligent transportation systems, 21(4), 1572-1583.

[15] Kim, J. U., Kwon, J., Kim, H. G., Lee, H., & Ro, Y. M. (2018, October). Object bounding box-critic networks for occlusion-robust object detection in road scene. In *2018 25th IEEE International Conference on Image Processing (ICIP)* (pp. 1313-1317). IEEE.

[16] Bui, H. M., Lech, M., Cheng, E., Neville, K., & Burnett, I. S. (2016, July). Using grayscale images for object recognition with convolutional-recursive neural network. In 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE) (pp. 321-325). IEEE.

[17] Mu, D., Sun, W., Xu, G., & Li, W. (2021). Random Blur Data Augmentation for Scene Text Recognition. *IEEE Access*, *9*, 136636-136646.

[18] Das, S., Tasnim, N., Shihab, M. I. H., Khan, M. F., & Mahmud, R. A. Team Passphrase: Dhaka AI Vehicle Detection.

[19] Boesch, G. (2022, January 17). *Object detection in 2022: The Definitive Guide*. viso.ai. Retrieved January 18, 2022, from https://viso.ai/deep-learning/object-detection/

[20] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

[21] Solawetz, J. (2021, September 21). Yolov5 new version - improvements and evaluation. Roboflow Blog. Retrieved January 18, 2022, from https://blog.roboflow.com/yolov5-improvements-and-evaluation/

[22] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).