Cross Platform Training of Neural Networks to Enable Object Identification by Autonomous Vehicles

by

Sangeet Sankaramangalam Ulhas

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2019 by the
Graduate Supervisory Committee:

Spring Berman, Chair
Kathryn Johnson
Sze Zheng Yong

ARIZONA STATE UNIVERSITY

August 2019

ABSTRACT

Autonomous vehicle technology has been evolving for years since the Automated Highway System Project [1]. However, this technology has been under increased scrutiny ever since an autonomous vehicle killed Elaine Herzberg, who was crossing the street in Tempe, Arizona in March 2018 [2]. Recent tests of autonomous vehicles on public roads have faced opposition from nearby residents [14]. Before these vehicles are widely deployed, it is imperative that the general public trusts them. For this, the vehicles must be able to identify objects in their surroundings and demonstrate the ability to follow traffic rules while making decisions with human-like moral integrity when confronted with an ethical dilemma, such as an unavoidable crash that will injure either a pedestrian or the passenger.

Testing autonomous vehicles in real-world scenarios would pose a threat to people and property alike. A safe alternative is to simulate these scenarios and test to ensure that the resulting programs can work in real-world scenarios. Moreover, in order to detect a moral dilemma situation quickly, the vehicle should be able to identify objects in real-time while driving. Toward this end, this thesis investigates the use of cross-platform training [15] for neural networks that perform visual identification of common objects in driving scenarios. Here, the object detection algorithm Faster R-CNN [25] is used. The hypothesis is that it is possible to train a neural network model to detect objects from two different domains, simulated or physical, using transfer learning. As a proof of concept, an object detection model is trained on image datasets extracted from CARLA, a virtual driving environment, via transfer learning [26]. After bringing the total loss factor to 0.4, the model is evaluated with an IoU metric [24]. It is determined that the model has a precision of

100% and 75% for vehicles and traffic lights respectively. The recall is found to be 84.62% and 75% for the same. It is also shown that this model can detect the same classes of objects from other virtual environments and real-world images. Further modifications to the algorithm that may be required to improve performance are discussed as future work.

DEDICATION

To my mother, father, brother and sister for their unconditional support.

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Dr. Spring Berman of the School for Engineering of Matter, Transport and Energy at Arizona State University. The door to Prof. Berman’s office was always open through her Google Calendar when I ran into trouble or had a question about my research or writing. She consistently encouraged this paper to ultimately be my own work and steered me in the right direction.

I would also like to thank the experts who were involved in the validation and survey for this research project: Dr. Kathryn Johnson, Dr. Sze Zheng Yong, and Mr. Shenbagaraj Kannapiran. Without their passionate participation and input, the validation and survey could not have been successfully conducted.

I would also like to acknowledge Mr. Hunter Larkins of the Department of Psychology at Arizona State University for all his help in completing the study part of this thesis, and I am gratefully indebted to Mr. Saurabh Prabhu and Mr. Akshay M Nair for their generous help and valuable comments on this thesis.

Finally, I must express my very profound gratitude to my parents and to my brother and sister for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Author

Sangeet Sankaramangalam Ulhas

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Background

Research spanning almost seven decades about human behavior around automation has shown that we have a natural tendency to trust reliable automated systems too much. After we relinquish our control to these machines for an extended period, it becomes second nature to our minds to trust them and to expect them to work as required. However, new research suggests that this trust may not extend to autonomous vehicles, also called self-driving vehicles, or vehicles with autopilot. For example, a recent study found that while drivers used the autopilot control feature to travel 34.8% of their miles, they maintained 'functional vigilance' while the autopilot was enabled [3]. Advanced driver assistance systems like automatic lane keeping, smart cruise control, and other technologies are now commonplace in most mid-range consumer automobiles. Some researchers claim that a few hundred thousand kilometers of driving experience for autonomous cars programmed with machine learning algorithms is sufficient to prepare the technology for widespread deployment [27]. However, many argue that the safety record for self-driving cars has not yet been proven [13]. While these autonomous vehicles would eliminate the incidence of fatal crashes that are due to human causes [4] (see Figure 1), public trust in autonomous vehicles is essential for their widespread adoption.

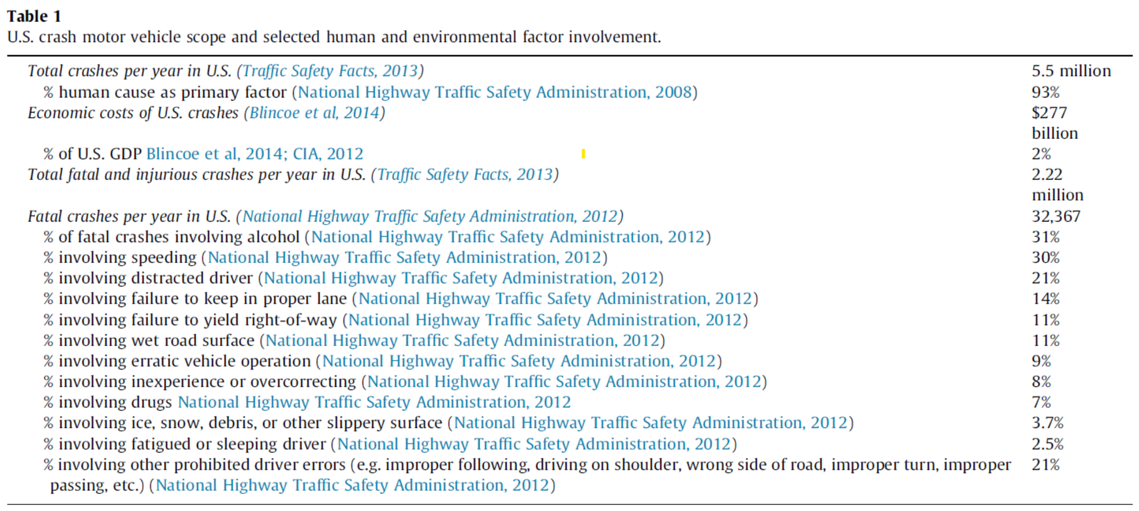
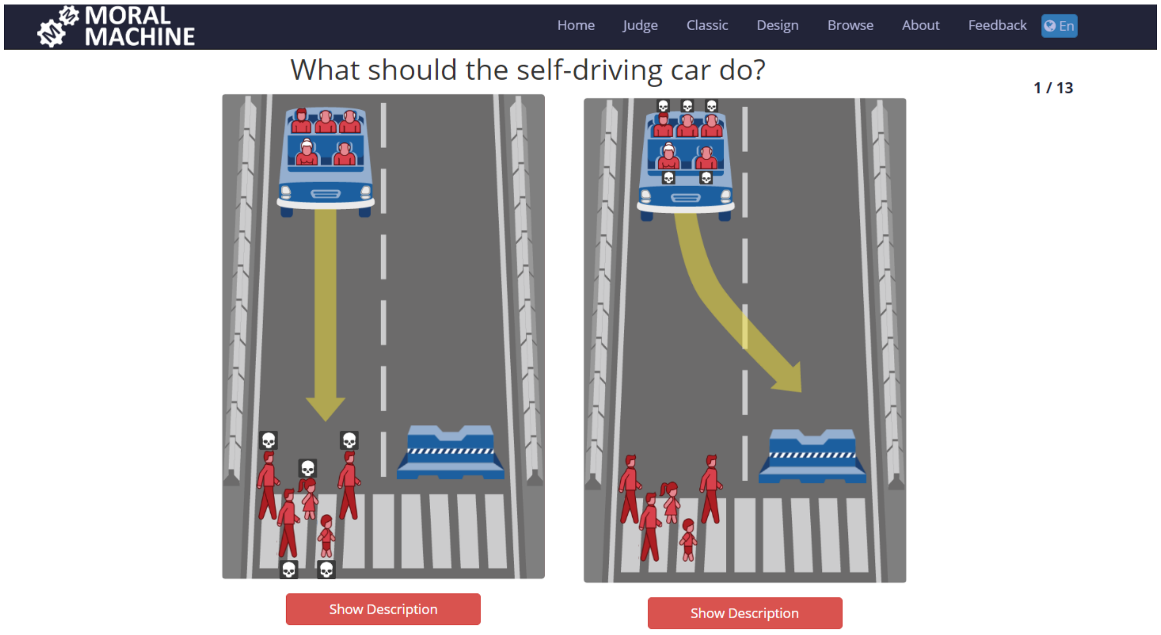


Table 1
U.S. crash motor vehicle scope and selected human and environmental factor involvement.

Total crashes per year in U.S. (Traffic Safety Facts, 2013)	5.5 million
% human cause as primary factor (National Highway Traffic Safety Administration, 2008)	93%
Economic costs of U.S. crashes (Blincoe et al, 2014)	$277 billion
% of U.S. GDP Blincoe et al, 2014; CIA, 2012	2%
Total fatal and injurious crashes per year in U.S. (Traffic Safety Facts, 2013)	2.22 million
Fatal crashes per year in U.S. (National Highway Traffic Safety Administration, 2012)	32,367
% of fatal crashes involving alcohol (National Highway Traffic Safety Administration, 2012)	31%
% involving speeding (National Highway Traffic Safety Administration, 2012)	30%
% involving distracted driver (National Highway Traffic Safety Administration, 2012)	21%
% involving failure to keep in proper lane (National Highway Traffic Safety Administration, 2012)	14%
% involving failure to yield right-of-way (National Highway Traffic Safety Administration, 2012)	11%
% involving wet road surface (National Highway Traffic Safety Administration, 2012)	11%
% involving erratic vehicle operation (National Highway Traffic Safety Administration, 2012)	9%
% involving inexperience or overcorrecting (National Highway Traffic Safety Administration, 2012)	8%
% involving drugs National Highway Traffic Safety Administration, 2012	7%
% involving ice, snow, debris, or other slippery surface (National Highway Traffic Safety Administration, 2012)	3.7%
% involving fatigued or sleeping driver (National Highway Traffic Safety Administration, 2012)	2.5%
% involving other prohibited driver errors (e.g. improper following, driving on shoulder, wrong side of road, improper turn, improper passing, etc.) (National Highway Traffic Safety Administration, 2012)	21%

Figure 1 – Human Factor Involvement in Crashes [4]

Although automation erases some of the limitations of human-operated machinery, it will still need to respond to difficult dilemmas entailing ethical and moral decisions. An example of such a problem that is relevant to autonomous vehicle decision-making is the “trolley problem”. A trolley problem is an analogy which represents a decision between hurting a few or many [28]. The MIT moral machine experiment gathered data from 40 million decisions on trolley-problem scenarios, presented online in ten languages, from people spanning over 233 countries and territories [12] (see Figure 2). The analysis of this data determined both global moral preferences and individual variations in preferences. This study revealed the factors that influenced their decisions. In addition, a recent study by Dr. Kathryn Johnson at Arizona State University (ASU) has isolated some of the moral values or factors behind decisions that determine a person’s driving behavior [23]. The two core values that really mattered were ‘power’ and ‘benevolence’. It is now apparent that autonomous vehicles will encounter ethical dilemmas, such as inevitable crash scenarios similar to the trolley problem. The technology available to us today can be used to tackle some of these challenges.



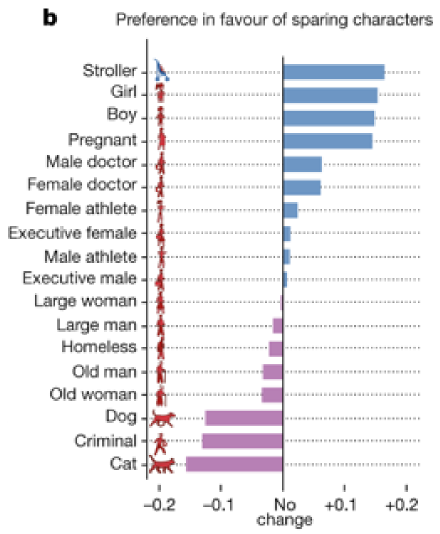


Figure 2 - MIT Moral Machine Experiment [12]

Towards this end, the work in this thesis is motivated by the problem of programming moral integrity into autonomous vehicles based on studies of human decision-making during simulated vehicle crashes. I conducted these studies, in collaboration with Dr.

Kathryn Johnson at ASU, using the CARLA driving simulator, which is described below in Section 1.3. In addition, experiments can be performed on object detection and lane tracking using a new version of a small-scale physical driving testbed called CHARTOPOLIS, developed at ASU [5][29], which includes small robots that emulate autonomous vehicles.

This thesis accomplishes the task of building a bridge that closes the gap in experiments conducted on the CHARTOPOLIS testbed and the simulation environment, CARLA, by demonstrating cross platform training of a neural network. The method used in this thesis can also be called a normalization technique (object classification knowledge becomes transferrable) in which a control system implemented on the small robot robotic vehicles on the physical testbed can be transferred to simulated vehicles in CARLA in order to check its functionality in scenarios that are much more complicated than ones that can be implemented on the testbed. For example, there are limitations to the realism of modeling pedestrians on the testbed. While we could train an object detection model, designed as described in this thesis, to detect a pedestrian on the testbed, there might be situations in which the pedestrian performs complex maneuvers that cannot be reproduced easily on the testbed. In such a scenario, the exact same object detection model could be used on CARLA which can simulate complex traffic scenarios. In this way, the object detection model serves as a cross-platform tool that could potentially bridge the gap between physical and simulated environments when conducting research on autonomous vehicle control strategies. In the following sections, we describe Unreal Engine, which is used to design the simulated environment; CARLA Driving Simulator, which is used for data collection and object detection; CHARTOPOLIS testbed, which is used as the physical domain to complement the simulation in CARLA; Object detection algorithms that are used in this thesis; TensorFlow library, which is used to deploy the object detection algorithms; Transfer learning and its applications in this thesis.

1.2 Unreal Engine

CARLA (see Section 1.3) was developed with a simulated world created using Unreal Engine 4. Unreal Engine is a graphics engine developed by Epic Games, first released in 1998 for the first-person shooter game 'Unreal'. Since then, it has undergone various iterations through development year after year and has become quite popular among game developers. The most recent version is Unreal Engine 4, released in 2014 [6]. For the purposes of building the driving simulator detailed in this thesis, Unreal Engine 4.22 was built on an Ubuntu distribution (16.04 LTS) of Linux. The following specification of the hardware is used.

Operating System	Ubuntu 16.04 LTS, 64-bit
Processor	10-Core Intel Xeon Processor
Memory	16 GB ECC x 2
GPU	Nvidia GeForce RTX 2080Ti

Table 1 – Hardware used to build CARLA

Other simulators like Autoware, Airsim, TORCS, Udacity Simulator, and the Donkeycar simulator were also considered for the simulated crash studies. From this list, CARLA and Airsim were identified as the best fits for the purposes of this thesis because they use Unreal Engine, which provides a lot of the functionality needed for simulation studies with acceptable fidelity and scalability. Another candidate for the graphics environment was Unity. Both Unity and Unreal Engine are capable of producing AAA quality graphics. They have extensive toolboxes that include a terrain editor, physics simulation, animation, advanced lighting, and virtual reality support. Both engines support 2D, and fully 3D rendered games. They run the latest rendering technologies including PBR (Physically Based Rendering), GI (Global Illumination), volumetric lights, post processing, and advanced shaders. While it is possible to produce the same simulation quality using either engines, Unity falls short when it comes to tools that are usable out of the box. Unreal provides presets that can be easily used and modified. This was apparent once CARLA was updated with its current sensor suite. For this reason, Unreal Engine was selected as the graphics engine in the simulation studies.

1.3 CARLA Driving Simulator

CARLA [30] is an open-source simulator for autonomous driving research that was developed by a group of developers with the help of its founding sponsors Intel Labs and the Computer Vision Centre at Universitat Autonoma de Barcelona, Barcelona to facilitate the development, training, and validation of autonomous vehicle systems. Not limited to open-source code, CARLA also provides open-source digital assets (urban layouts, buildings, vehicles). The simulator provides a wide array of sensor suites and, environmental conditions, full control of all static and dynamic agents, map generation, and many more features. [30] CARLA has two primary modules, the simulator module, and the Python API module. The former will be referred to as the *server-side* in this thesis. This module performs most of the graphic-intensive processes and hence requires a dedicated GPU to run satisfactorily. With the Python API, we can control all static and dynamic agents in the simulation, attach sensors to the agents, and record/reproduce all the data generated using these sensors. CARLA is also well-integrated with the Robot Operating System (ROS). The ROS bridge that is being developed by CARLA's developers can facilitate the programming of multiple agents with a composite moral profile that is built from the results of human subject studies using the driving simulator. These programs can then be adapted to work with the CHARTOPOLIS testbed.

1.4 CHARTOPOLIS testbed

CHARTOPOLIS [29] is a miniature testbed that is being designed as a laboratory for testing human-robot interaction in a scale model of an urban traffic environment. It has small car shaped mobile robots called Go-CHARTS that has all the sensory capabilities of an autonomous vehicle. Along with carefully designed roads with proper lane systems that conform with the driving regulations in the United States (to scale), it also has artistically designed buildings, a working traffic regulation system with traffic signs and signals and model pedestrians. CHARTOPOLIS provides a safe environment in which it is possible to test our future autonomous vehicle control strategies, such as those based on a composite moral profile from human subject studies. It is also part of the vision that this thesis work proposes – to be able to train a single neural network that can perform various tasks like object detection and vehicle navigation across multiple domains/platforms. In this thesis, the proposed domains are the CHARTOPOLIS testbed (physical domain) and the CARLA driving simulator (virtual domain).

1.5 Object Detection Algorithms

Within the field of computer vision, various algorithms based on deep learning techniques have been developed for detecting particular objects in images and videos. These algorithms are now quite widely used and have numerous applications besides autonomous vehicle driving, including people counting, face detection, medical diagnosis, gaming, and security. There are three algorithms that are used widely in the industry [17]:

- Faster R-CNN
- YOLO
- SSD

In this thesis, we make use of Faster R-CNN, which is a recent incarnation of region-based convolutional neural networks (R-CNN). Although the first R-CNN algorithms were computationally expensive, they have now been improved and can achieve real-time rates using very deep neural networks [16]. R-CNN extracts regions of interest in the shape of boxes from an input image by using selective search. It then checks whether any of these boxes contains an object. The regions that contain objects are extracted first. For each of these regions, a CNN is used to extract particular features. Finally, these features are used to detect objects. Unfortunately, R-CNN is relatively slow due to the multiple steps involved in the process [17].

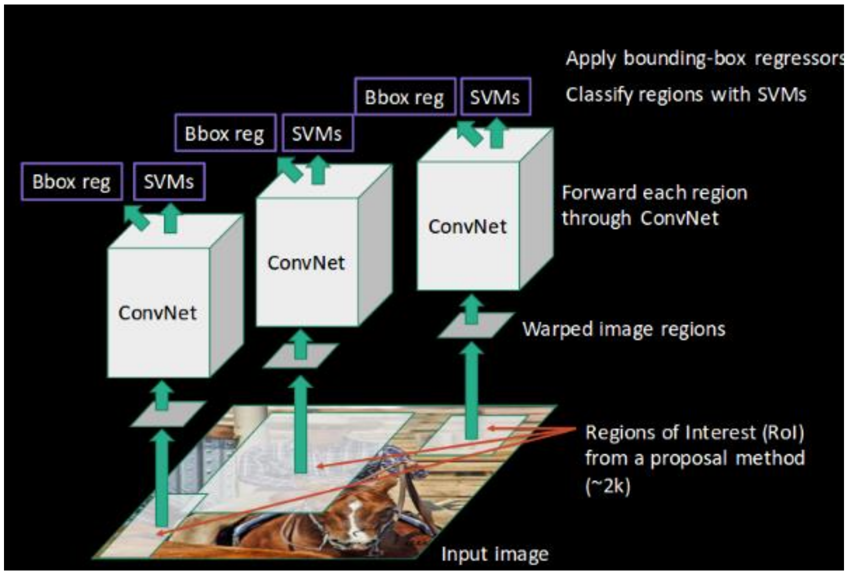

Apply bounding-box regressors
Classify regions with SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Figure 3 – Illustration of the R-CNN Object Detection Algorithm [17]

A different version of the algorithm, Fast R-CNN, passes an entire image to ConvNet (short for Convolutional Neural Networks), which then generates regions of interest instead of passing the extracted regions from the image. Instead of using three different models as in R-CNN, the algorithm uses a single model that extracts features from the regions, classifies them into different classes, and returns the bounding boxes of object classes under investigation. These three steps are done simultaneously, enabling Fast R-CNN to execute faster than R-CNN. However, Fast R-CNN is not fast enough when applied on a large dataset, since it also uses selective search for extracting the regions.

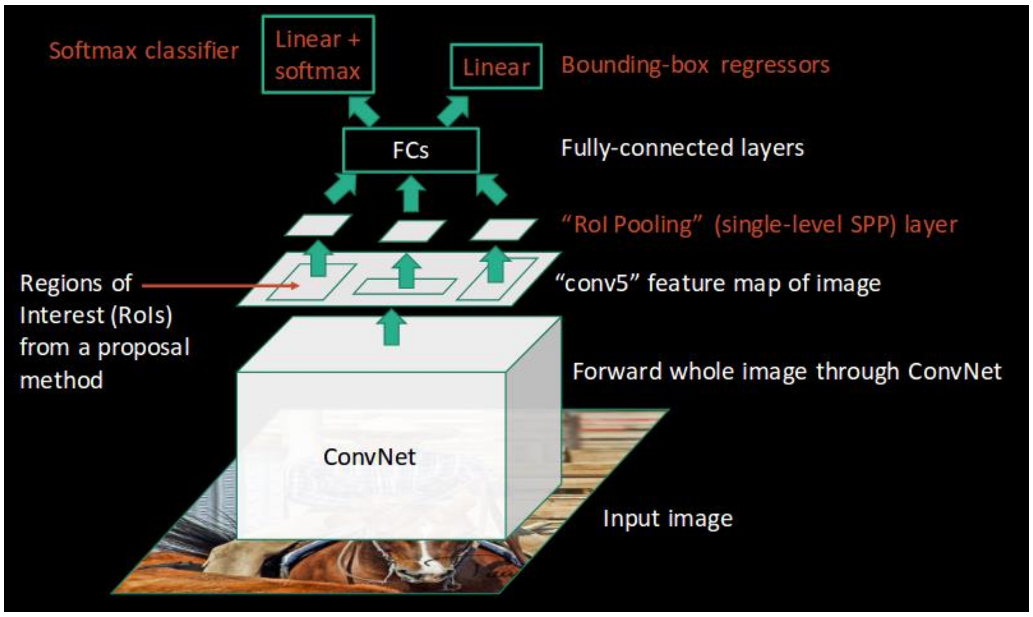

Softmax classifier
Linear + softmax
Linear
Bounding-box regressors
FCs
Fully-connected layers
"RoI Pooling" (single-level SPP) layer
Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image

Figure 4 – Illustration of the Fast R-CNN Object Detection Algorithm [17]

The iteration of the algorithm that we use, Faster R-CNN, fixes this problem. It replaces selective search with the region proposal network (RPN) technique. Feature maps are extracted from the input image using ConvNet, and the maps are passed through an RPN, which then returns the object proposals. Finally, the maps are classified, and bounding boxes are predicted. Object detection algorithms like Faster R-CNN are possible means to identify potential traffic hazards on the two platforms proposed in this thesis work, the CHARTOPOLIS testbed and the CARLA driving simulator. Alternatives like YOLO, SSD or RFCN can also be used based on experimental requirements of speed and accuracy. Faster R-CNN was used in this thesis work due to its ease of access when it comes to overcoming common debugging hurdles because of better documentation. It also yielded better accuracy when a minimally trained neural network performed object detection.

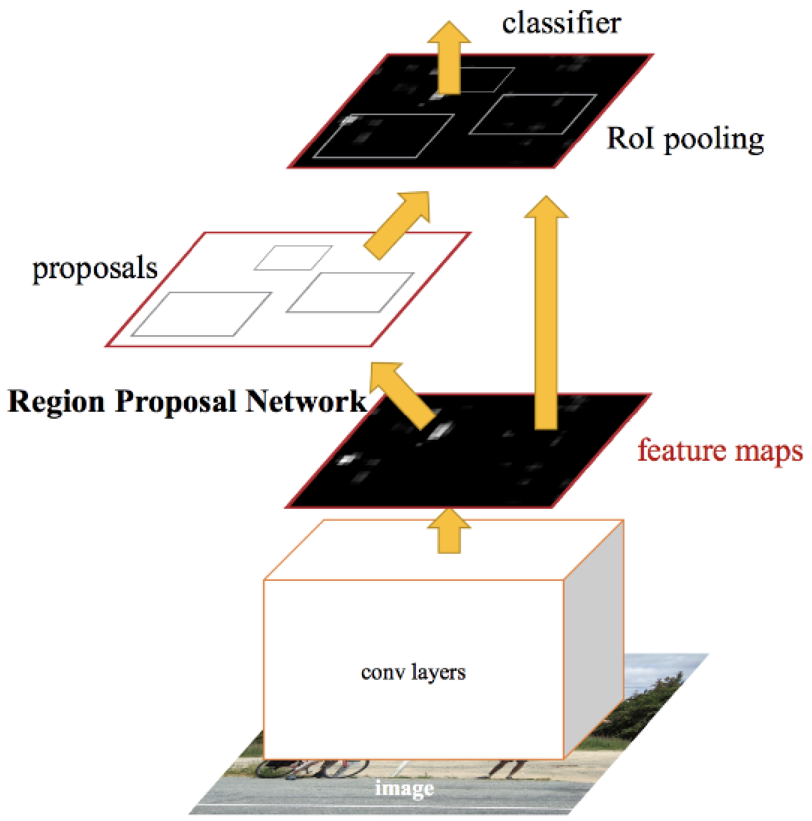


Figure 5 – Illustration of the Faster R-CNN Object Detection algorithm [17]

1.6 TensorFlow

TensorFlow is an open-source software library for dataflow and differentiable programming across a wide range of applications. It was developed by Google initially for internal use. Later, it was released under the Apache License 2.0. It is a symbolic math library and is used for machine learning applications [7]. It can also be utilized as an interface for expressing machine learning algorithms through the implementation for executing such algorithms. The primary advantage that it provides is that it can be run across various platforms on different types of systems using the same algorithms. It is being used for conducting research in more than a dozen disciplines of computer science, including speech recognition, natural language processing, computer vision, robotics, data mining, computational drug discovery, and geographic information extraction.

1.7 Transfer Learning

Transfer learning is a machine learning method in which a neural network model that was trained for a particular task is then repurposed by additional training for an entirely different application. This method saves time on model training and produces better results in test and validation of the checkpoint model (intermediate model output or save point) created by training a pre-trained model on a new dataset of the object class under investigation. It is popular for applications that can benefit from the use of pre-trained models [8]. For example, knowledge (specific intrinsic features recognized by the neural network) realized by training a model to detect cars could then be used to train the same model to detect trucks. Domain adaptation can be used when we are required to learn from a source data distribution and subsequently create a model that performs well on a different (but related) target data distribution. Since the objective of this thesis is to show that minimal training of a neural network will suffice to enable cross-platform training of the network for object identification in autonomous driving applications, transfer learning is the most essential part of this project. The performance of a model that is trained using transfer learning can exhibit three main improvements over its performance without the use of transfer learning [18]: a higher start, a higher slope, and a higher asymptote (see Figure 6). A successful application of transfer learning will exhibit all these benefits. However, it

should be noted that in general, it is not obvious that there will be a benefit to using transfer learning in a particular domain until after the model has been developed and evaluated. The use of pre-trained neural network models in transfer learning is limited only by the researcher's imagination. A model may be downloaded and used out of the box, which is the approach that we take in this work. Further, the model can be used as a feature extraction model. Here, the output of the model from a layer prior to the output layer of the model is used as input to a new classifier model [19]. This means that we make use of the intrinsic features that were captured by the pre-trained neural network while we train it further on a separate dataset.

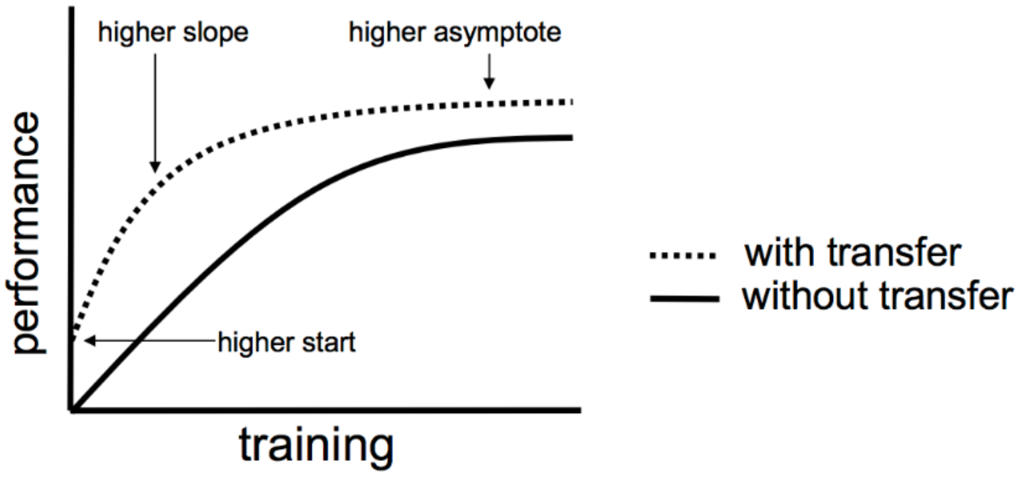
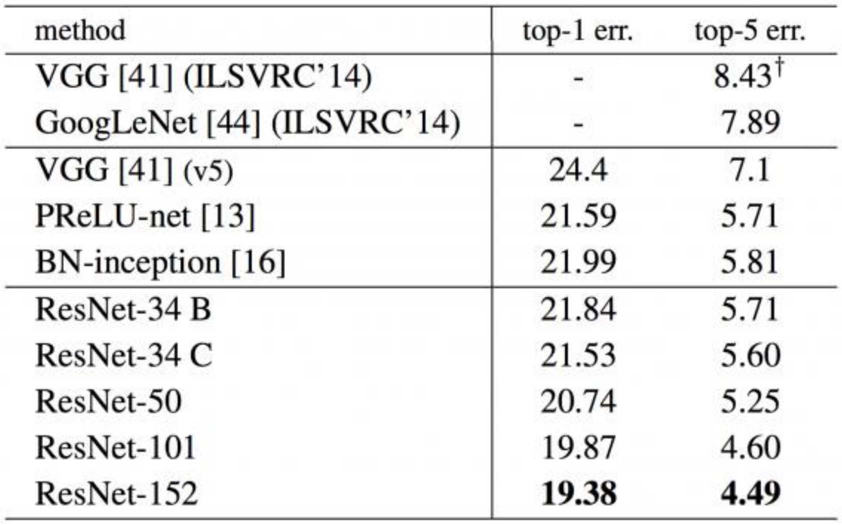


Figure 6 - Benefits of Transfer Learning [18]

Deep CNNs extract low, middle, and high-level features and classifiers in an end-to-end multi-layer fashion. When a deep neural network starts to converge on an asymptote (as the loss decreases), a degradation problem occurs: the deeper the network is, the faster its accuracy becomes saturated, after which its performance degrades rapidly. The cause for this is neither overfitting nor the addition of more layers causing higher training error but poorly optimizable loss functions. This deterioration is proof that not all systems are easily optimizable. As a solution, Microsoft introduced the deep residual learning network (ResNet). Error rates of single-model results on the ImageNet validation set [31] are listed in Figure 7. The figure shows that ResNet is the better choice since it results in the lowest errors. In the case of top-1 score (see Figure 7), you check if the top class (the one having the highest probability) is the same as the target label. In the case of top-5 score, you check if the target label is one of your top 5 predictions (the 5 ones with the highest probabilities). The 50-layer ResNet (ResNet-50 in Figure 7) is constructed by replacing each 2-layer block with a 3-layer bottleneck block. This model requires 3.8 billion FLOPs. The 101-layer and 152-layer ResNets requires 7.6 billion and 11.3 billion FLOPs, respectively. Because of these high computational requirements, choosing a network with a large number of hidden layers would defeat our objective of achieving accurate object detection with quick, minimal training of the model.



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

Figure 7 - Error Rates on ImageNet Validation Set [22]

To further motivate the use of transfer learning in this thesis, let us look at an example problem. Imagine that we have a pre-trained object detection model that can detect cats and dogs from images. These are four-legged mammals that share similar visual characteristics which can be recognized by the hidden layers in the pre-trained model. Now imagine that we need to create an object detection model that can detect horses. It is much harder to train a model with a small image dataset of horses to reach the same level of performance as a model that is modified from cat-dog detection through further learning (transfer learning). A study indicates that we reduce the size requirement of the new dataset by implementing transfer learning [20]. This simply means that we will need fewer pictures of horses to train the new model. To summarize, transfer learning enables the use of a much smaller image dataset to train and test a neural network that can detect particular object classes across multiple domains (in our case, the physical testbed CHARTOPOLIS and the virtual environment CARLA). The application of transfer learning can also vastly improve the final performance of the neural network as shown in Figure 6. These are the primary motives behind the use of transfer learning in this thesis work.

CHAPTER 2

DRIVING SIMULATION STUDY

We used the driving simulator CARLA in a study, designed in collaboration with Dr. Kathryn Johnson at ASU, that sought to understand and differentiate the moral profiles of drivers based on study participants' decisions in a simulated "trolley-problem" scenario and their responses to a survey afterward. The details of this study are described in Immanuella Kankam's M.S. thesis, "Design of an Immersive Virtual Environment to Investigate How Different Drivers Crash" [9]. Participants viewed the simulation on three adjacent computer monitors and drove through the virtual environments, from the perspective of the driver, using a Logitech steering wheel (G920 Driving Force) and pedals. During the simulation, they would unexpectedly encounter three different inevitable crash scenarios, in which they could not avoid a collision and could only decide where to steer their vehicle, and therefore, where to crash. Snapshots of the three simulated scenarios are shown in Figures 8, 9, and 10 below. As described earlier, a wide range of virtual driving environments can be simulated in CARLA, which can be used to train and validate the object identification algorithms that are proposed in this thesis.

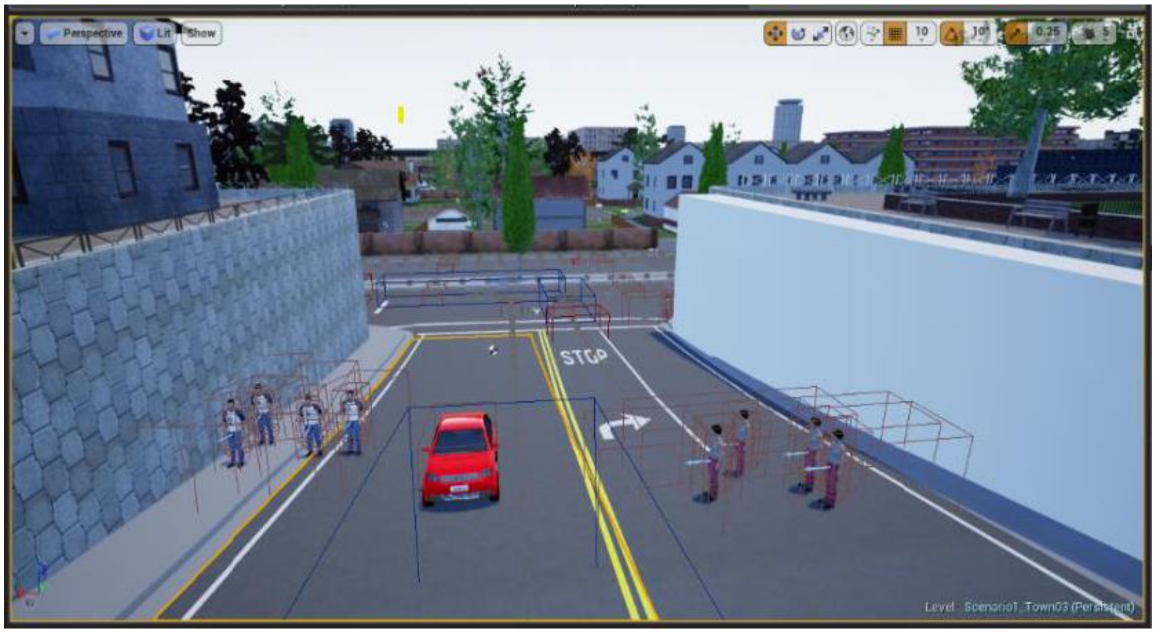


Figure 8 – Scenario 1

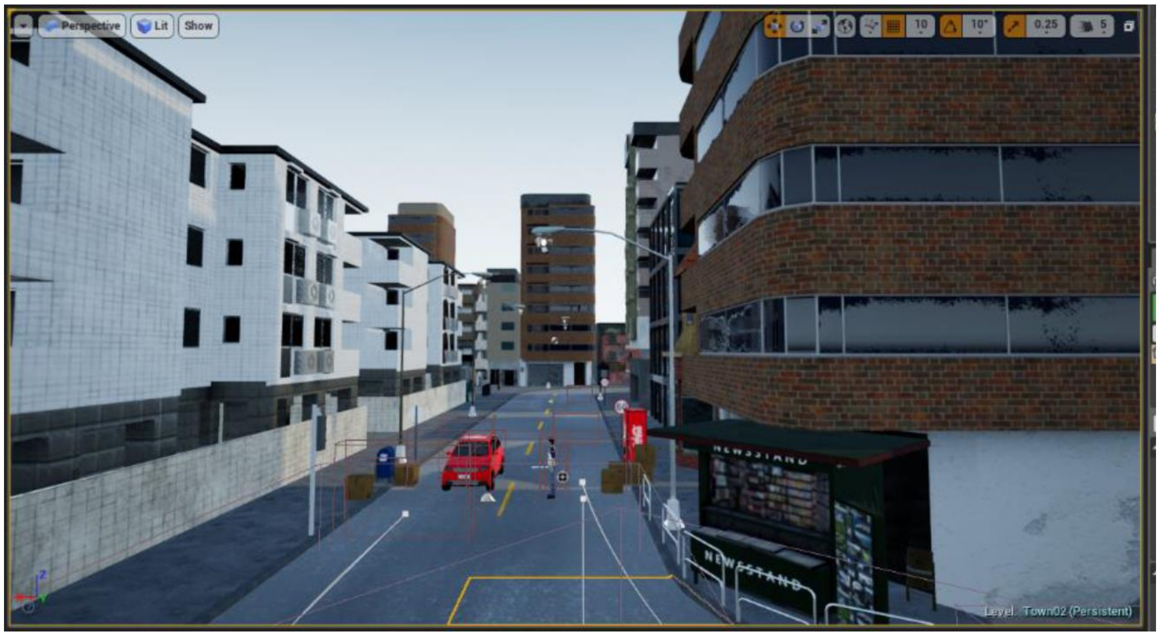


Figure 9 – Scenario 2

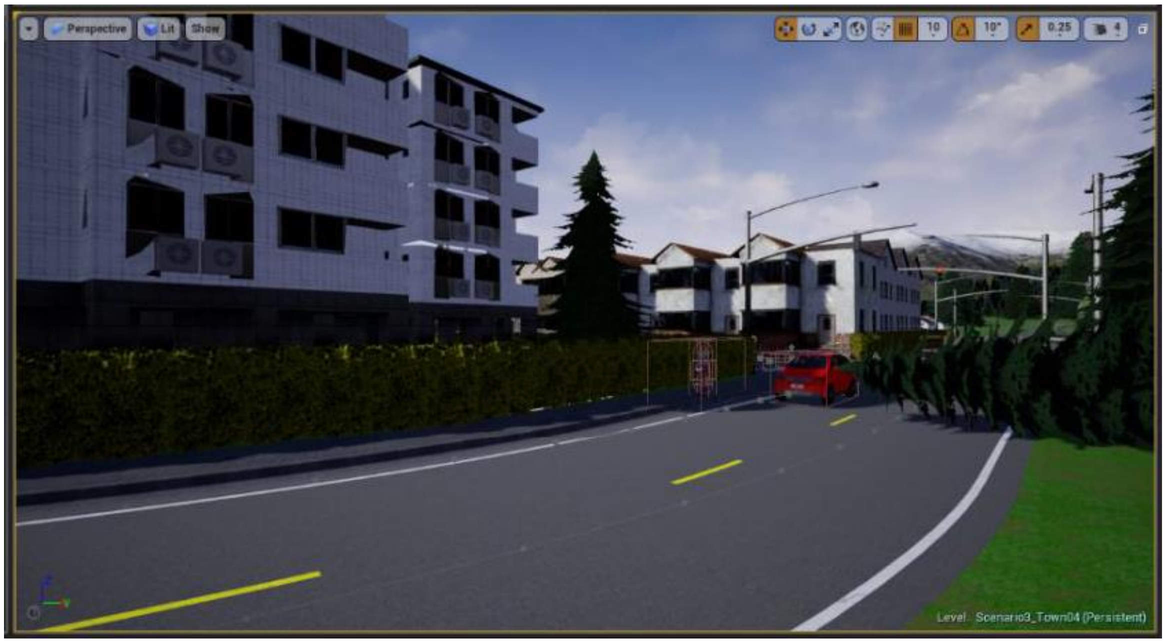


Figure 10 – Scenario 3

The driving environments in Figures 8, 9, and 10 were modeled in Unreal Engine 4 built on an Ubuntu distribution of Linux, as described in Section 1.2. These scenarios were then simulated in CARLA by using a Python API to call functions in Unreal Engine, which in turn uses an RPC (Remote Procedure Call) client, as illustrated in Figure 11.

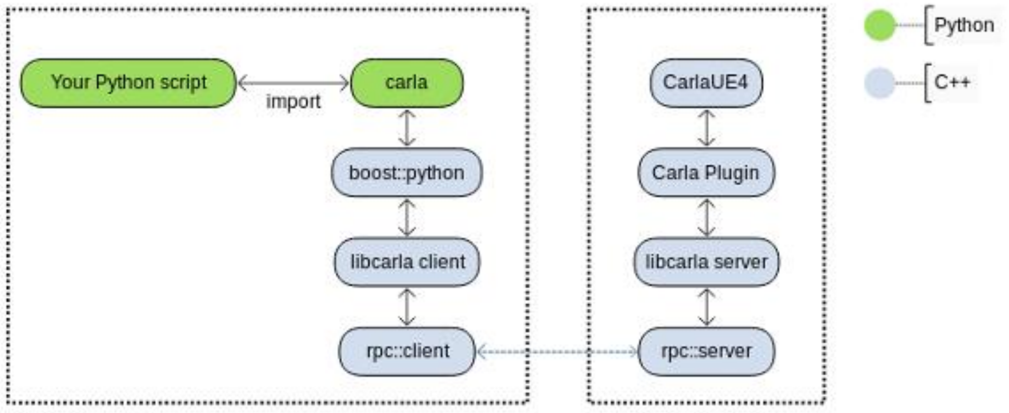


Figure 11 – RPC Server-Client

There were several major drawbacks in the initial implementation of these simulation studies. We identified solutions to these drawbacks, described below:

- Motion sickness: The simulations were displayed to the study participants on a three-monitor arrangement that required rapid lateral eye movement, which caused motion sickness in some participants. This was fixed by replacing the three-monitor setup with an ultra-wide curved monitor and better anti-aliasing.
- Low simulation framerates: The low frame rates of the simulation can be increased by running the Unreal Engine server-side on a different computer station and running only the client-side on Python, which is relatively less computationally intensive on a computer. The setup for the server-side processing is currently under development.
- Steering wheel sensitivity: The cause of extreme steering-wheel sensitivity was identified as a low input range from -1 to 1 that is being employed by the CARLA developers for inputs from steering wheels. A fix for this issue is also under development with help from CARLA's developers.

After the three-monitor setup was replaced with a curved monitor (see Figure 13) to reduce the incidence of motion sickness, a second group of participants used the simulator to conduct similar tests on a fourth scenario (with a similar situation), shown in Figure 12.

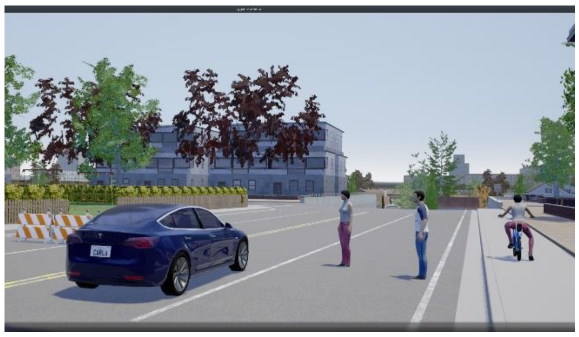


Figure 12 – Scenario 4

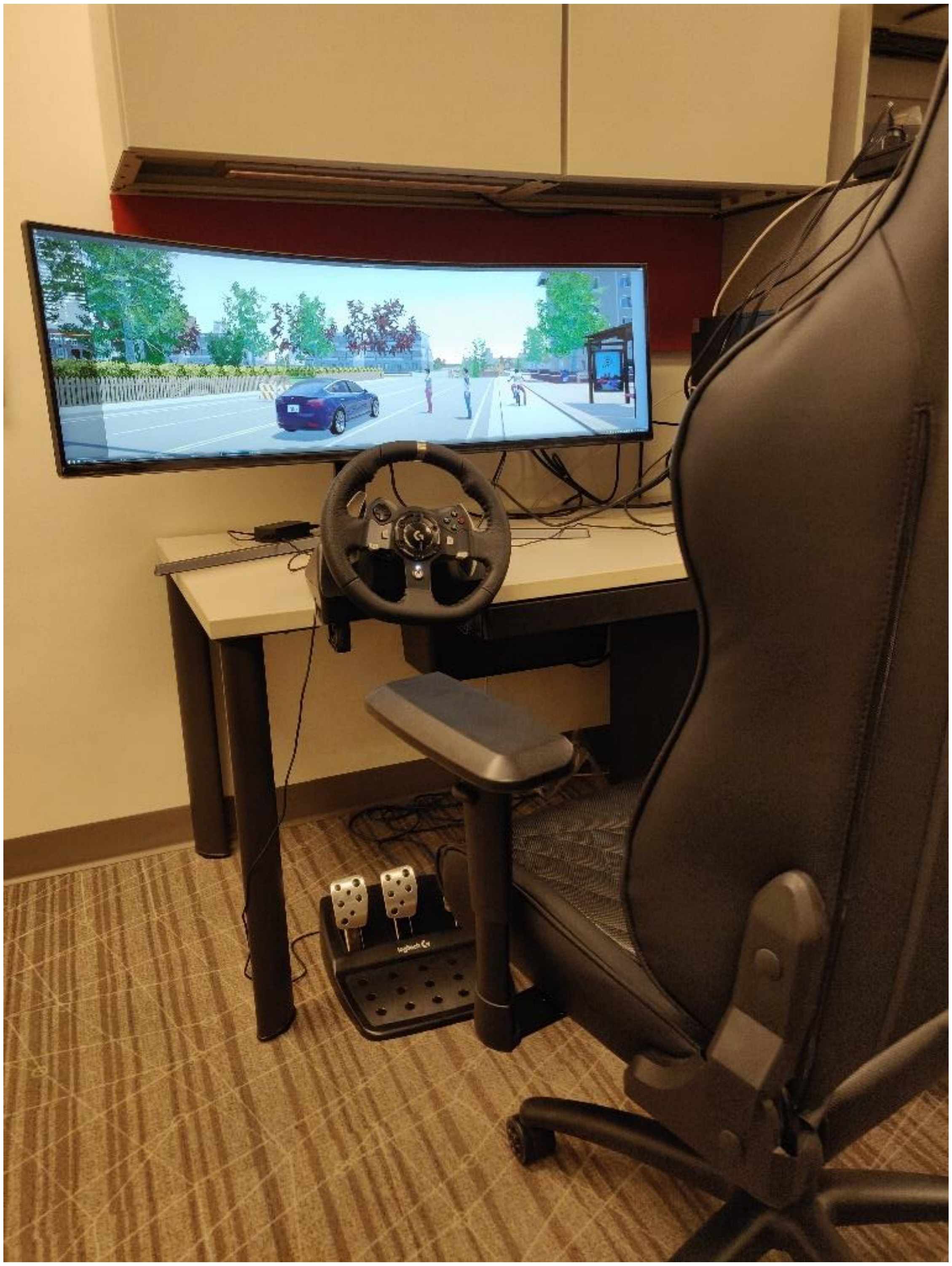


Figure 13 – Driving Simulator Setup

CHAPTER 3

DATASET COLLECTION METHODOLOGY

The paper "Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks" explains the need to explore our ability to train neural networks on data obtained from virtual worlds [10]. In this paper, they say, "*We demonstrate that a state-of-the-art architecture, which is trained only using these synthetic annotations, performs better than the identical architecture trained on human-annotated real-world data, when tested on the KITTI data set for vehicle detection*". The work in this thesis is inspired by a similar motivation. Moreover, we aim to evaluate our trained object detection model on images from the real world and other graphic engines. This would ultimately demonstrate that cross-platform training can be used to enable a neural network to detect objects on the physical CHARTOPOLIS testbed. The paper [10] used an image dataset obtained from the video game GTA-V and used its GPU's stencil-buffer data, which includes scene depth and other auxiliary information from the game for image segmentation. This aids in proper semantic segmentation and eliminates the need for human supervision during the segmentation process. When using image datasets from CARLA, this technique can be replaced by data from the array of sensors pre-programmed into the simulation environment. These sensors are the following:

1. Camera - RGB

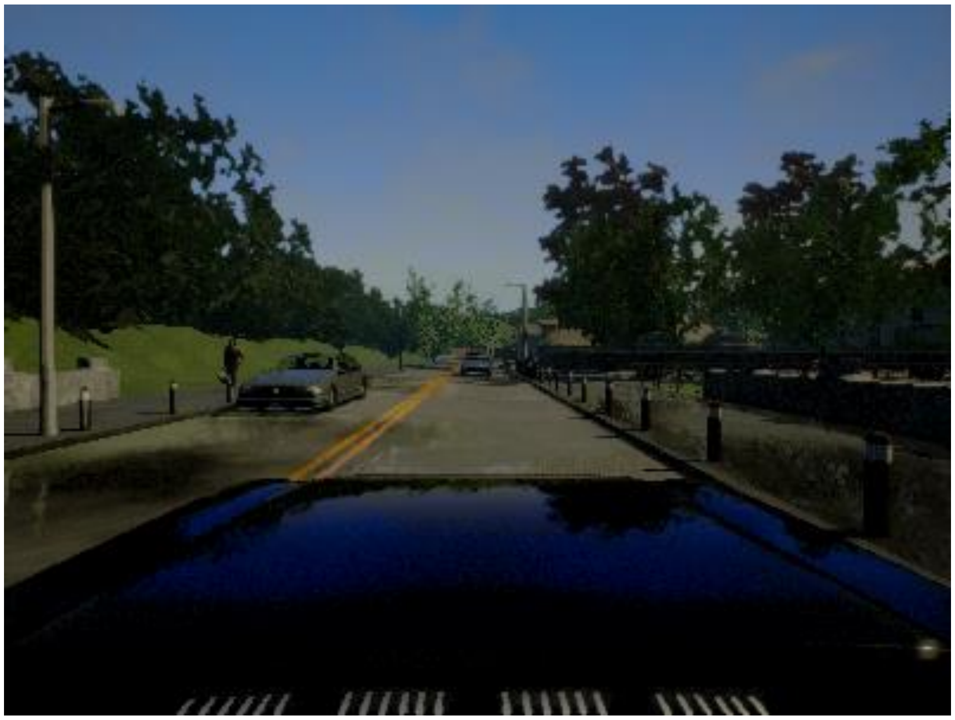


Figure 14 – Image from RGB Camera

2. Camera – Depth

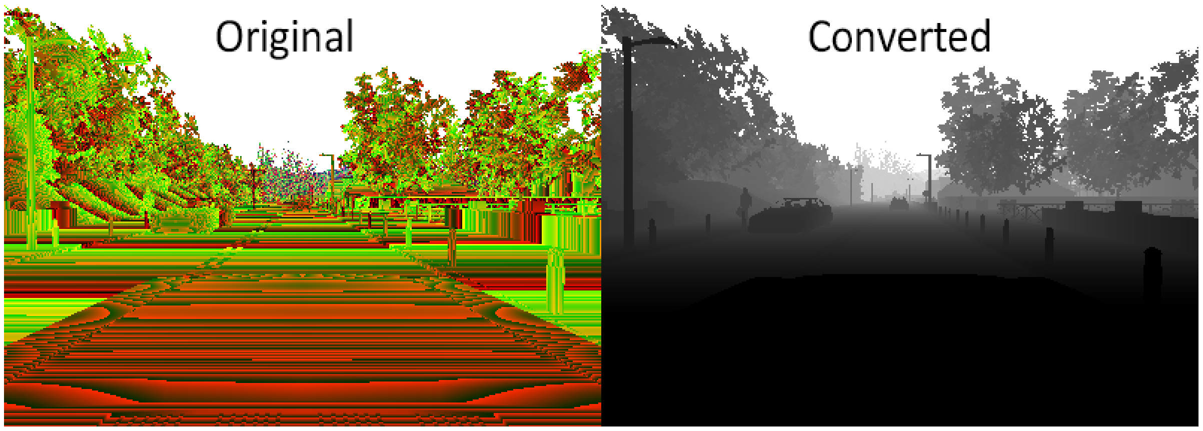


Figure 15 – Visualized Output of Depth Sensor

3. Camera – Semantic Segmentation

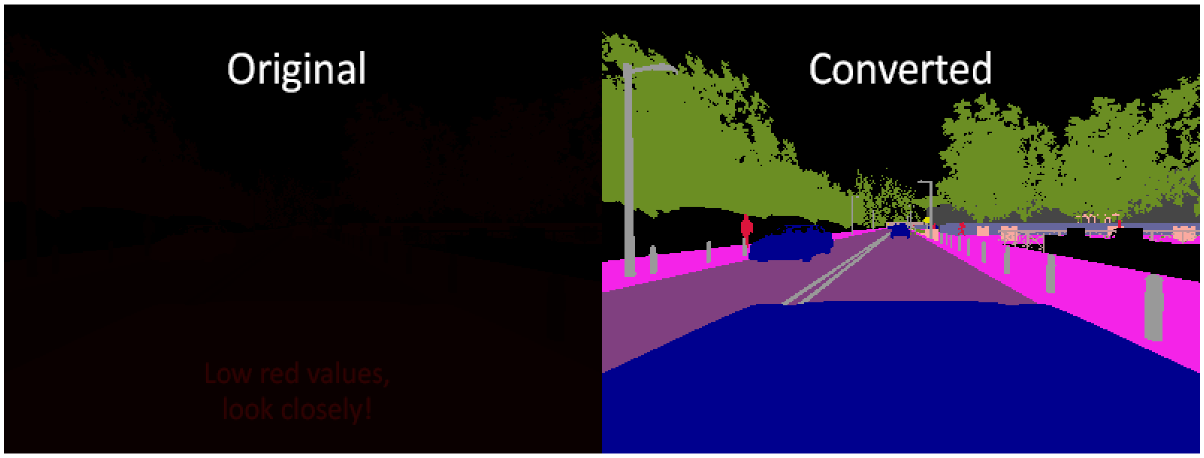


Figure 16 – Semantic Segmentation

4. Lidar – Ray cast

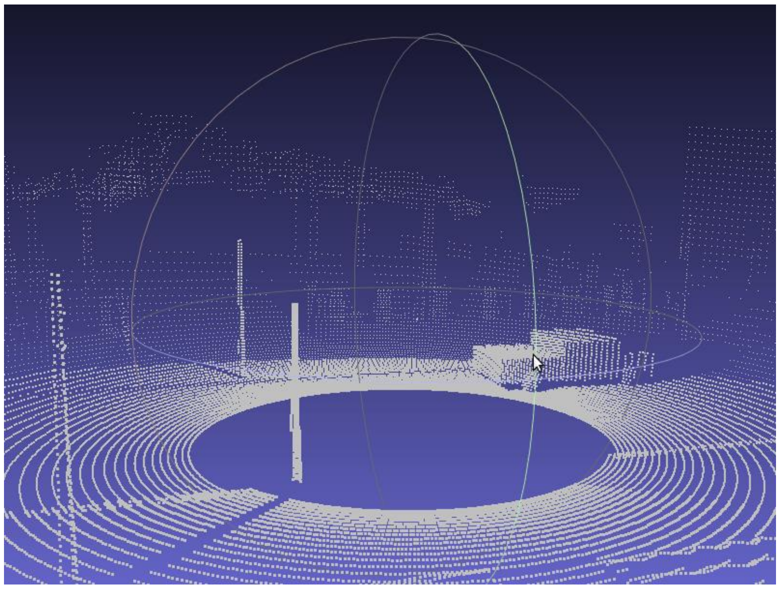


Figure 17 – LIDAR -Ray Cast

The in-situ dataset recording tools in CARLA make it easy to simulate different scenarios and create an annotated dataset of objects that are commonly present in driving environments. We created a training dataset with 820 images from CARLA simulations and labelled this dataset with the following object classes:

- vehicle
- bicycle
- motorbike
- traffic light
- traffic sign

A test dataset consisting of 208 similar images (from CARLA) was also created. Labelling was done manually using ‘labelImg’ (a tool used for annotation), as shown in Figure 18 [11].

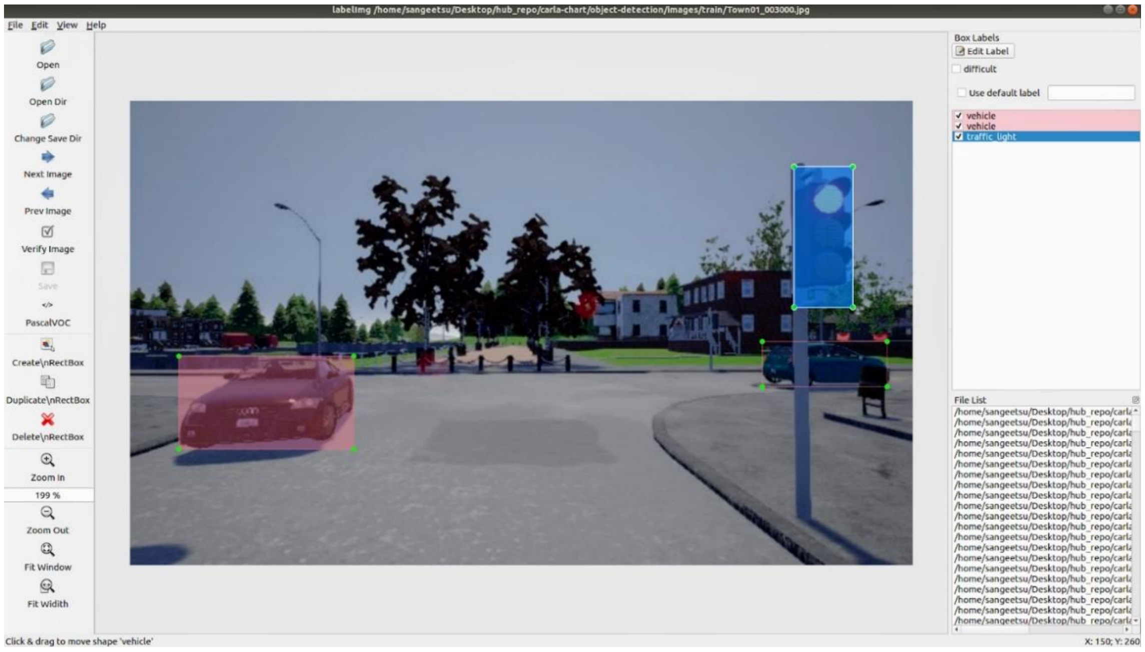


Figure 18 – Annotating a Sample Image Using LabelImg

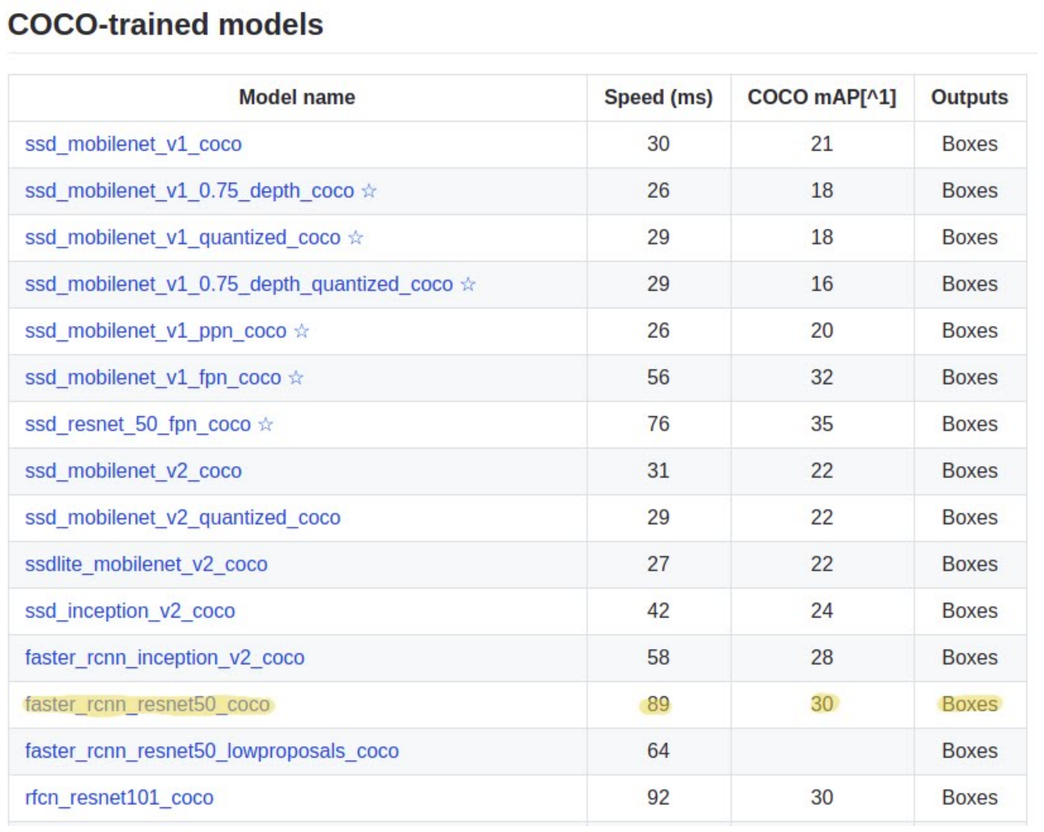
The ResNet-50 model was used to implement transfer learning. This model was pre-trained on the COCO (Common Objects in Context) dataset [32], which contains some of the object classes that are used in this thesis. Faster R-CNN, described in Section 1.5, was used as the object detection algorithm.

CHAPTER 4

OBJECT DETECTION TESTING AND RESULTS

4.1 Object Detection Algorithm Testing and Evaluation

Faster R-CNN, though slow, is very accurate. The speed versus accuracy of the model compared to some of the other models from the 'object detection model zoo' is shown in Figure 19.

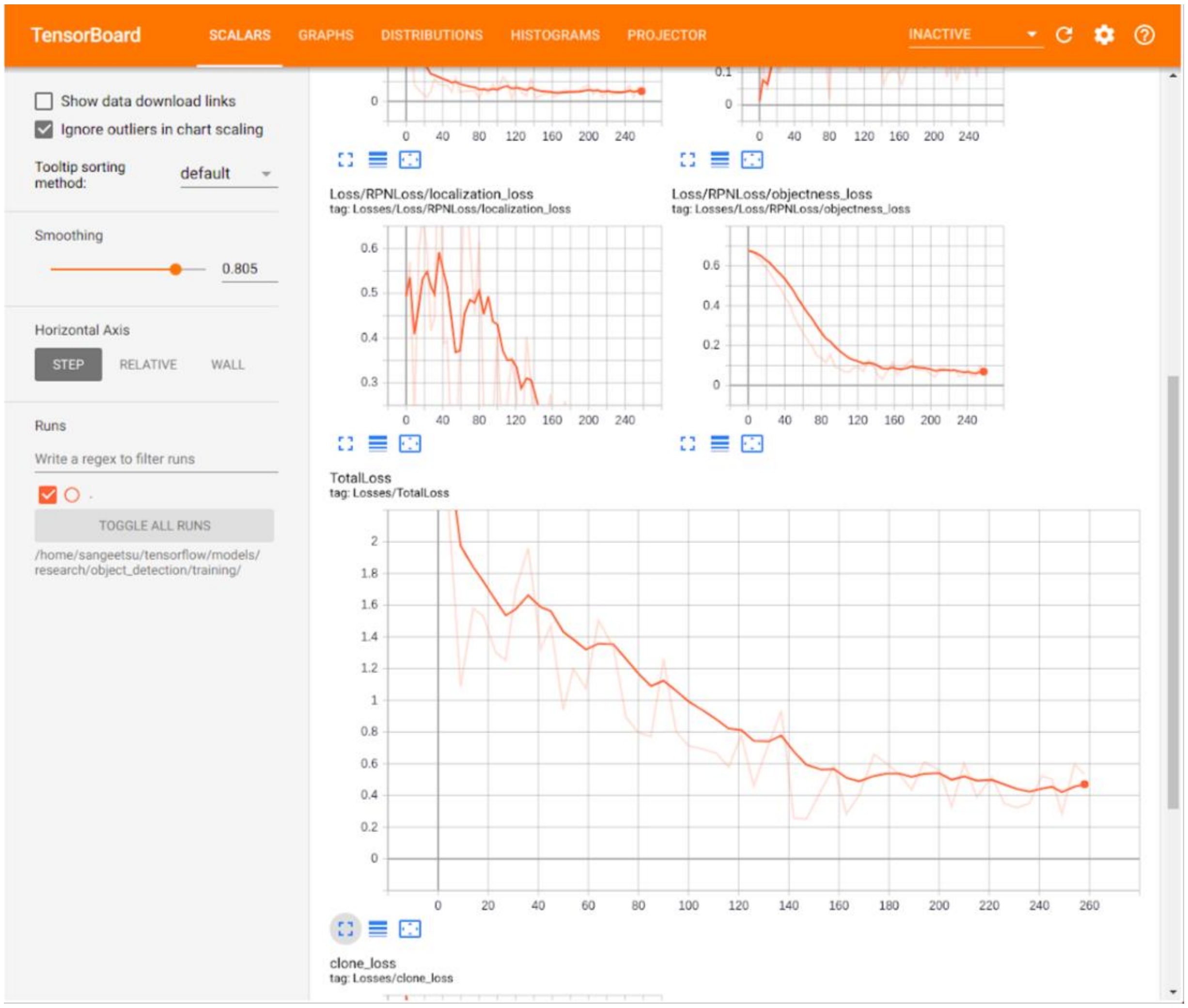


COCO-trained models

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes

Figure 19 – Object Detection Model Zoo

The newly created dataset is then used to do transfer learning, and the object detection algorithm is subsequently tested and validated as explained above. The training was terminated, and a checkpoint was created at 260 epochs. The total loss values decreased from 3 to below 0.5 in this time frame. The step number, although low, yielded acceptable recognition of images from different graphic engines and several real-world images. The graph of total loss versus epoch number is shown in Figure 20.



☐ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.805

Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs

Write a regex to filter runs

/home/sangeetsu/tensorflow/models/research/object_detection/training/

Figure 20 – Loss Curves from Tensorboard Data

The inference graph is then frozen, and the metadata stored for continuing training with a multiple GPU setup. As a proof of concept, the saved object detection model is then tested on ten images: five images from the recorded dataset, three real-world images, and two images from video games rendered using a different graphical engine (Rockstar Advanced Game Engine).

To evaluate the instance segmentation provided by the trained model, we use a method called Intersection over Union (IoU) for object detection. This is an evaluation metric that evaluates the predicted bounding boxed versus the annotations used for training (ground truth).

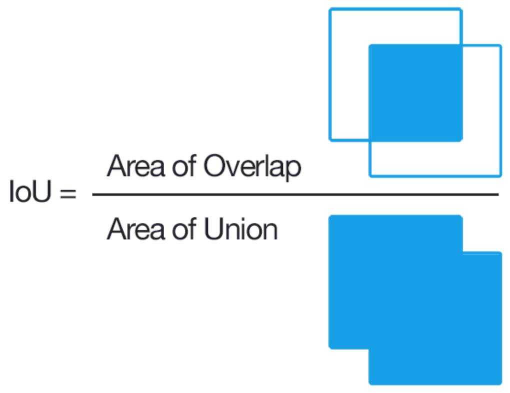


Figure 21 - Illustration depicting the IoU metric

In the numerator, we compute the area of overlap between the predicted bounding box and the ground-truth bounding box. The denominator is the area of the union or the area encompassed by both the predicted bounding box and the ground-truth bounding box. Here we see the evaluation of some of the images from the test dataset. An extensive list is attached in Appendix-C. The IoU values have been printed along with the bounding boxes on the top left corner.

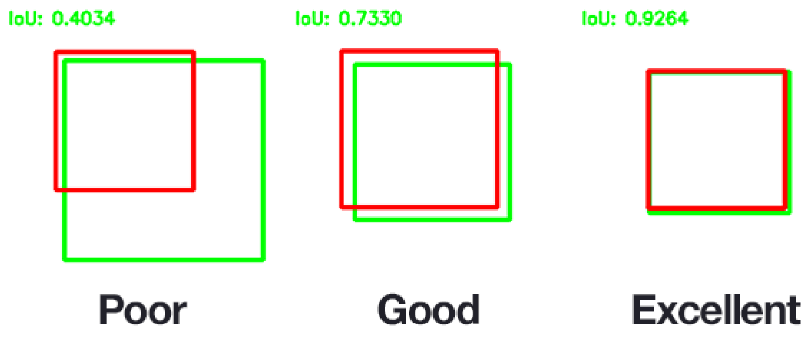


Figure 22 - An example of computing IoU for various bounding boxes

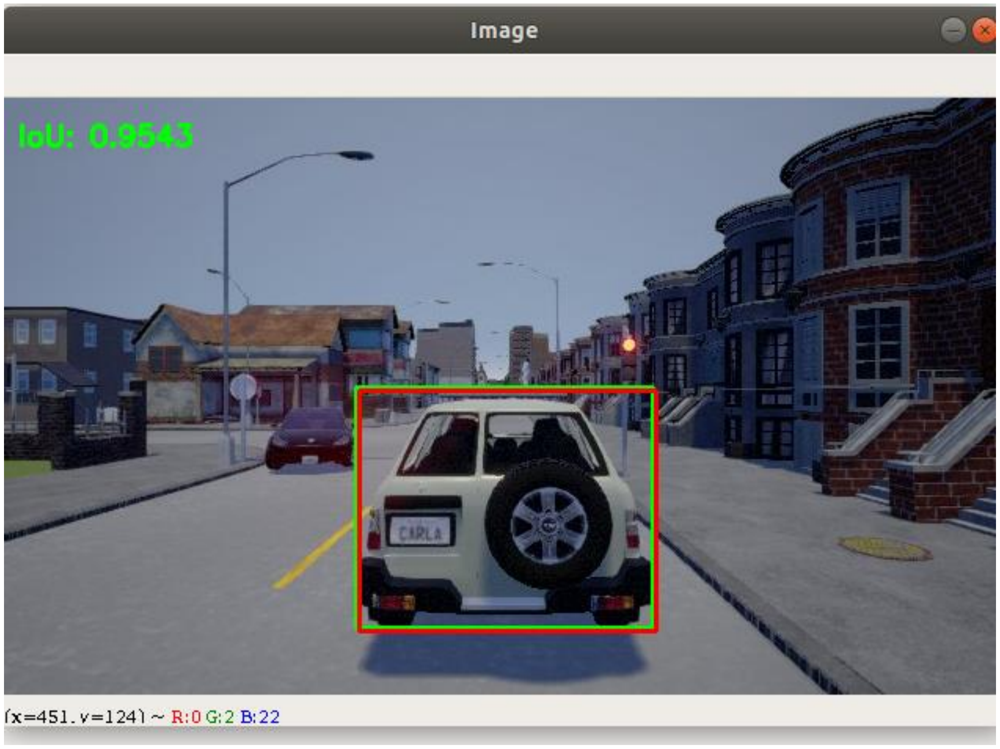


Figure 23 - IoU for car detection

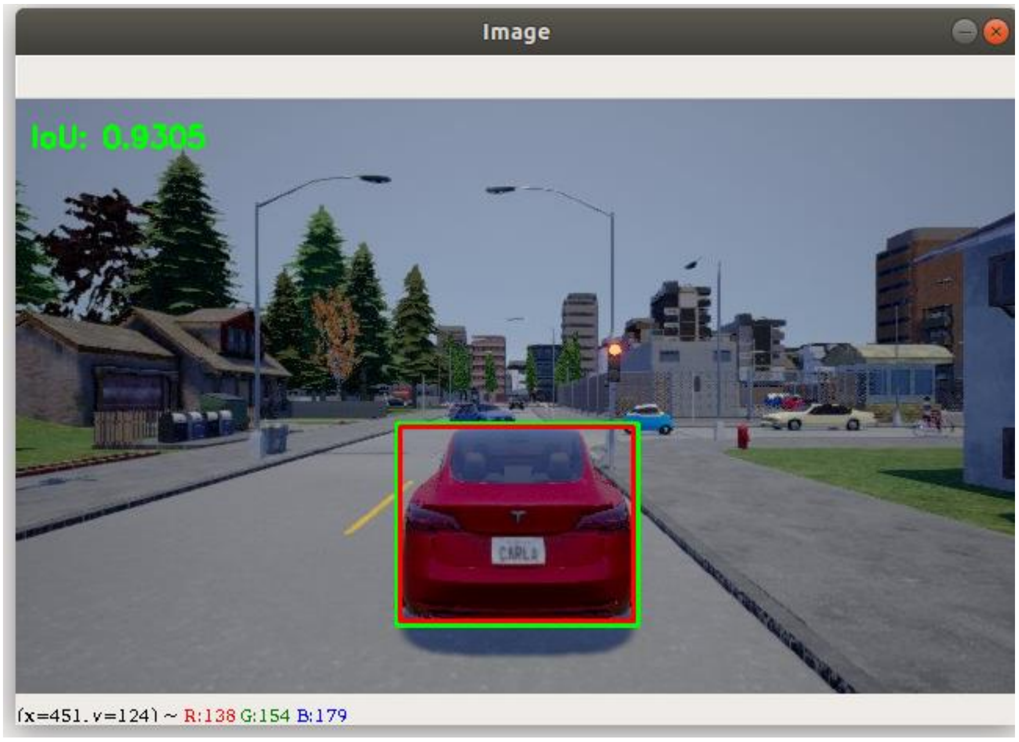


Figure 24 - IoU for car detection

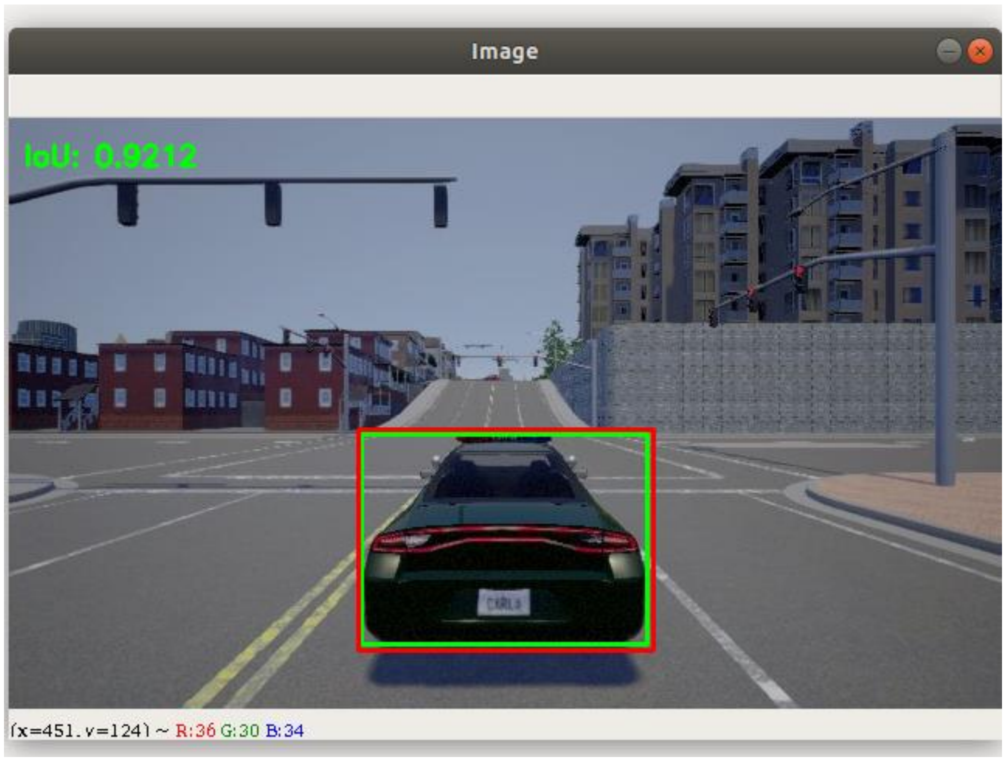


Figure 25 - IoU for car detection

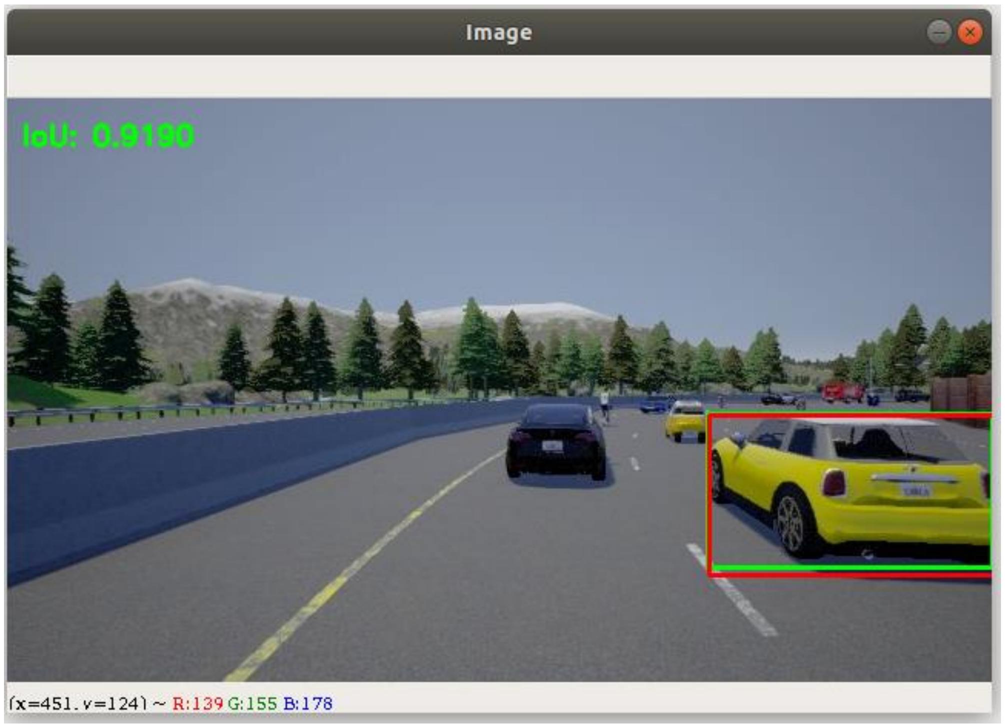


Figure 26 - IoU for car detection

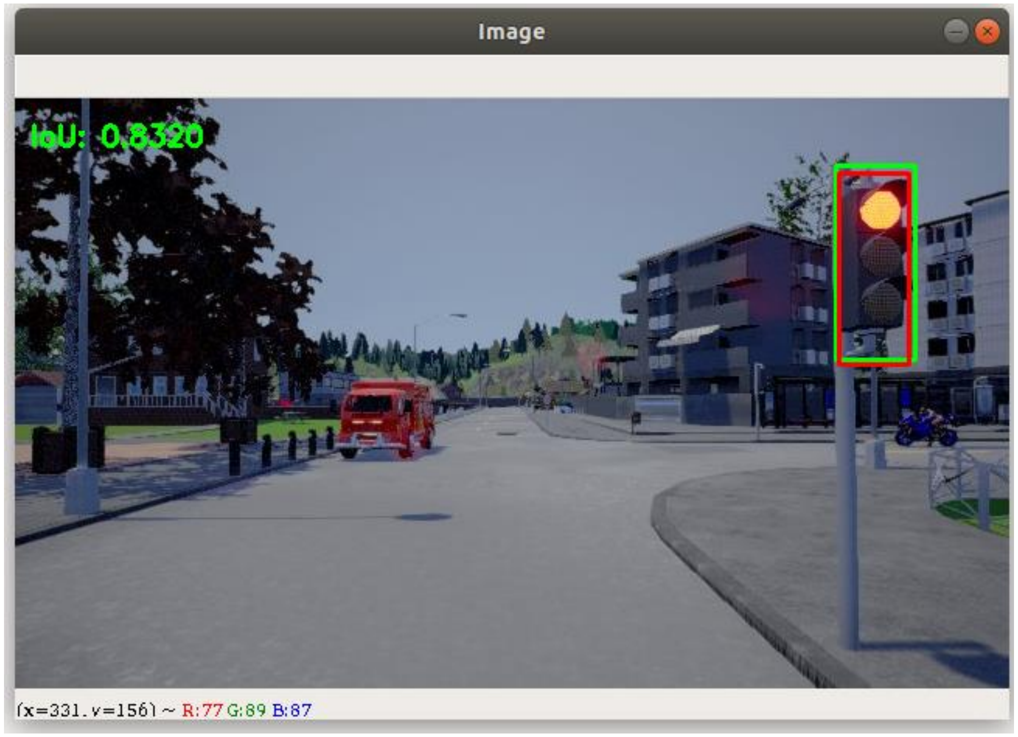


Figure 27 - IoU for traffic light detection

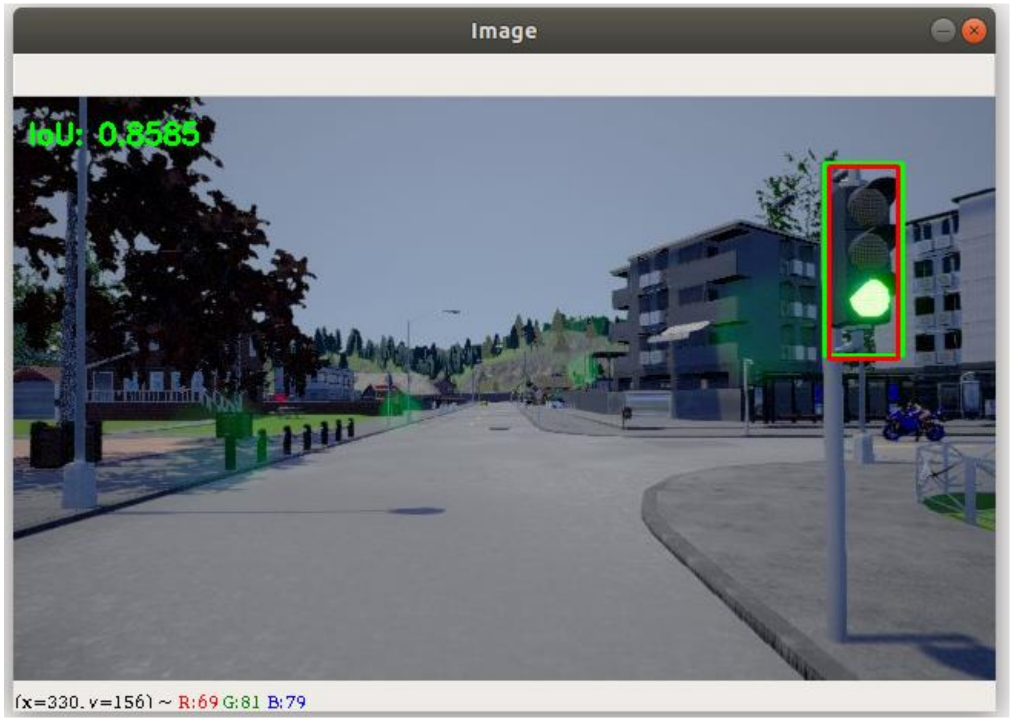


Figure 28 - IoU for traffic light detection

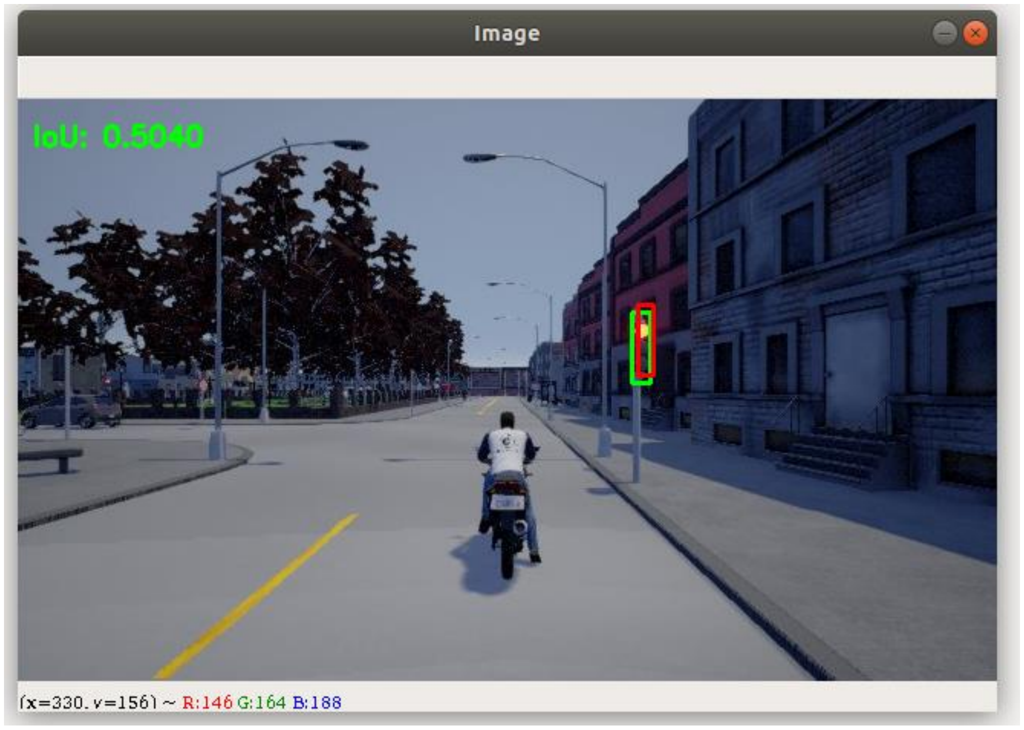


Figure 29 - IoU for traffic light detection

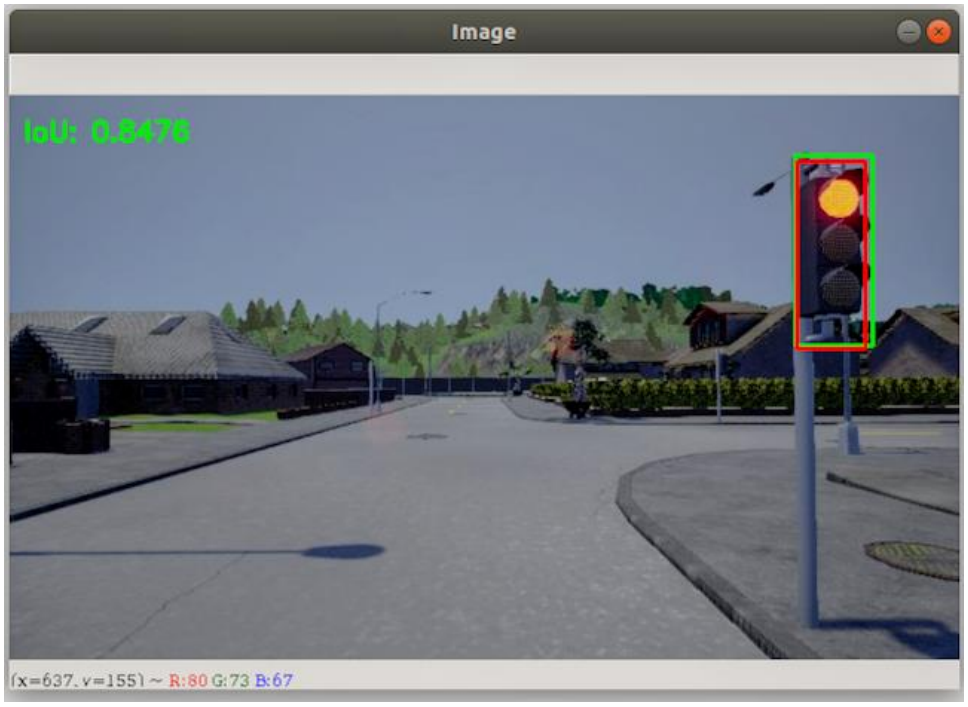


Figure 30 - IoU for traffic light detection

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Figure 31 - Precision and recall to measure accuracy of object detection [21]

In the field of information retrieval, *precision* is the fraction of retrieved documents that are relevant to the query. In our context, precision is a measure of how many detections of all the detections pertaining to an object class are correct. *Recall* is the fraction of relevant documents that are successfully retrieved. In our context, recall is a measure of how many were successfully detected out of the total number of images of a particular object class. Since the objective of this thesis does not require perfect accuracy scores, accuracy calculations are based on the IoU values in Figures 23-30. For calculating Precision and Recall, as with all machine learning problems, we have to identify True Positives, False Positives, True Negatives and False Negatives. To identify True Positives and False Positives, we use IoU. Using IoU, we now have to identify whether the detection (a Positive) is correct (True) or not (False). The most commonly used threshold is 0.5 - i.e., if IoU > 0.5, then it is considered a True Positive, and otherwise it is considered a False Positive. The COCO evaluation metric [32] recommends measurement across various IoU thresholds, but for simplicity, we will use a threshold of 0.5, which is the PASCAL VOC metric [33]. For calculating Recall, we need the count of Negatives. Since every part of the image where we did not predict an object is considered a negative, measuring "True" negatives is not that useful. So, we only measure “False” Negatives, i.e., the objects that our model has failed to identify.

Another factor that is taken into consideration is the confidence that the model reports for every detection. By varying our confidence threshold, we can change whether a predicted box is a Positive or Negative. Basically, all predictions (Box + Class) above the threshold are considered Positive boxes and all below it are Negatives. For every image, we have ground truth data that tells us the number of actual objects of a given class in that image. We calculate the IoU with the ground truth for every positive detection box that the model reports. Using this value and our IoU threshold (set to 0.5), we calculate the number of correct detections (A) for each class in an image. This is used to calculate the Precision for each class [TP/(TP+FP)].

Object class	**Image**	**IoU**	**TP**	**FP**	**Confidence**	**Actual No.**	**detected**	**FN**
Vehicle	Figure23	0.9543	TP		99%	2	1	1
Vehicle	Figure24	0.9305	TP		99%	4	3	1
Vehicle	Figure25	0.9212	TP		99%	1	1	0
Vehicle	Figure26	0.919	TP		99%	6	6	0
Traffic Light	Figure27	0.832	TP		85%	1	1	0
Traffic Light	Figure28	0.8585	TP		70%	1	1	0
Traffic Light	Figure29	-	-		0%	1	0	1
Traffic Light	Figure30	0.8476	TP		85%	1	1	0

Table 2 - Object detection precision-recall calculation

From Table 2, Precision (vehicle) = TP/(TP+FP) = 4/4 = 1 (100%);

and Precision (traffic light) = TP/(TP+FP) = 3/4 = 0.75 (75%)

Since we already have calculated the number of correct predictions (A) (True Positives) and the number of missed detections (False Positives), we can now calculate the Recall (A/B) of the model for that class using the formula TP/(TP+FN). The actual object detection results for this test dataset is shown in Appendix-C.

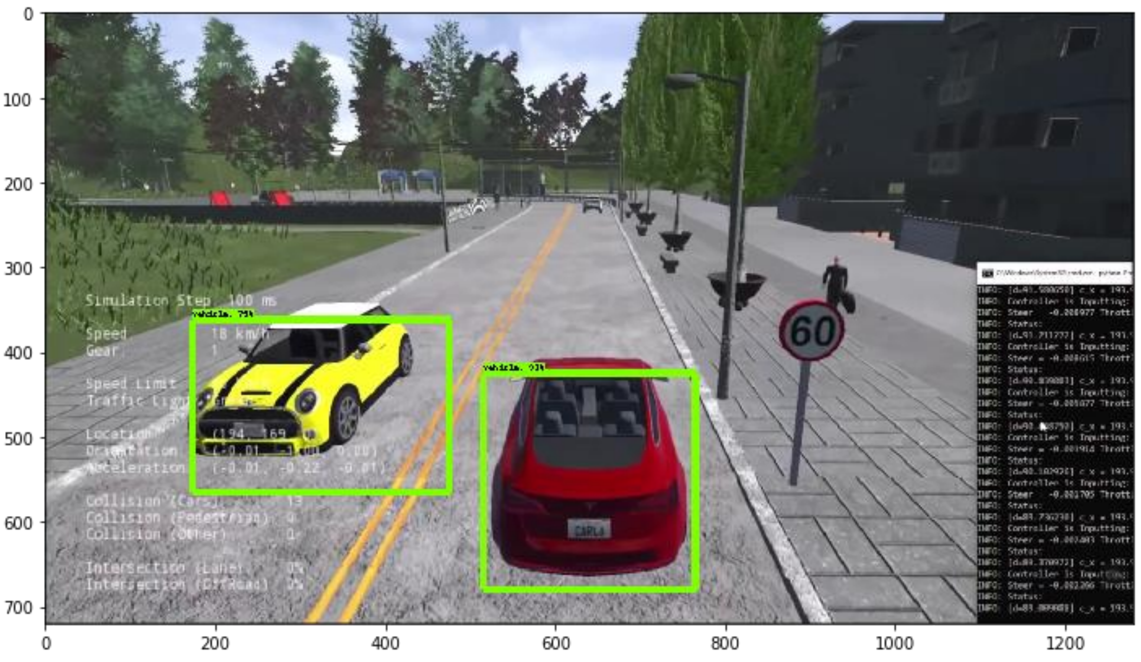
Recall (vehicle) = TP/(TP+FN) = 11/13 = 0.8462 (84.62%)

Recall (Traffic Light) = TP/(TP+FN) = 3/4 = 0.75 (75%)

These error rates would be unacceptable in a real-life autonomous vehicle. The accuracy of detection and other deep learning functions can be improved by using a larger image dataset to train the neural network. Use of more modern object detection algorithms like the DenseNet and pretraining such a neural network on a larger dataset that consists of video streams could significantly improve accuracy to yield desired results.

4.2 Test Results

The object detection algorithm was tested on the images in Figures 32-41, which display the pixel scaling on its sides and detected object classes as bounding boxes with their names and confidence levels in percentages. It can be seen from the results that the 260-epoch training helped the neural network to detect vertical traffic lights (similar models) and vehicles without fail. The classes that were not detected in environments alien to CARLA (like the horizontal traffic lights from Figure 33 which are rare in the model towns in CARLA) have not been detected in the images from CARLA either. Thus, ruling out the possibility of any interference from transfer learning, our work lays the foundation to build a cross platform trained neural network that will provide a useful tool for validating autonomous vehicle controllers in scenarios that are dangerous to implement in the real world, such as situations that present a moral dilemma



Simulation Step 100 ms

Speed 18 km/h
Gear 1

Speed Limit
Traffic Light

Location (194, 169
Orientation (-0.01, -1.00, 0.00)
Acceleration (-0.01, -0.22, -0.01)

Collision (Cars) 13
Collision (Pedestrian) 0
Collision (Other) 0

Intersection (Lane) 0%
Intersection (OffRoad) 0%

Figure 32 - Test Result

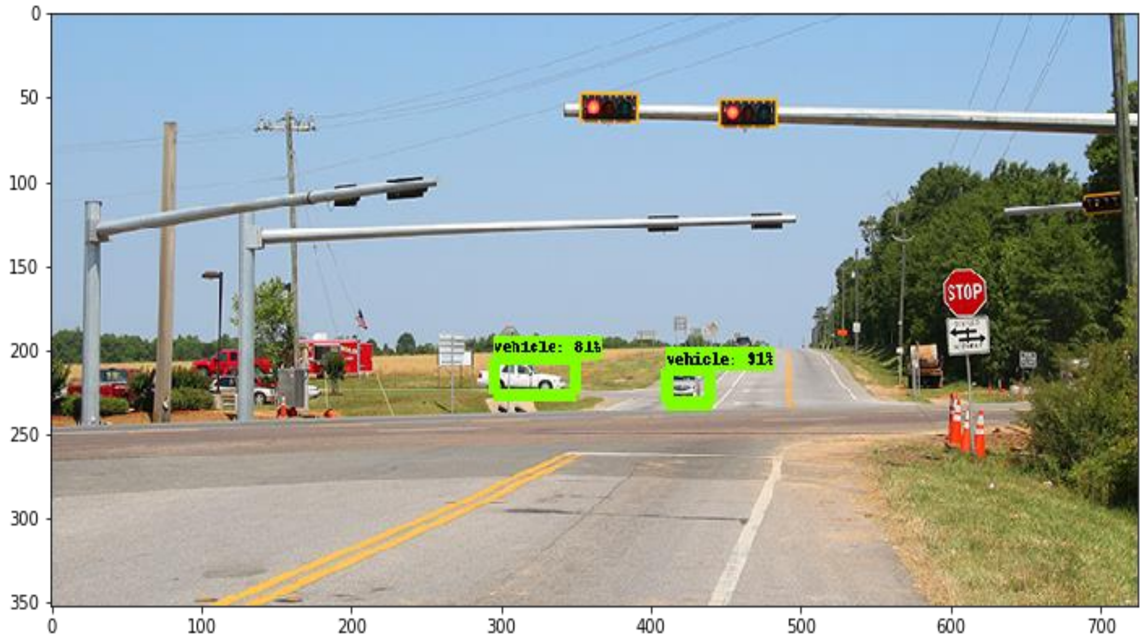


Figure 33 - Test Result

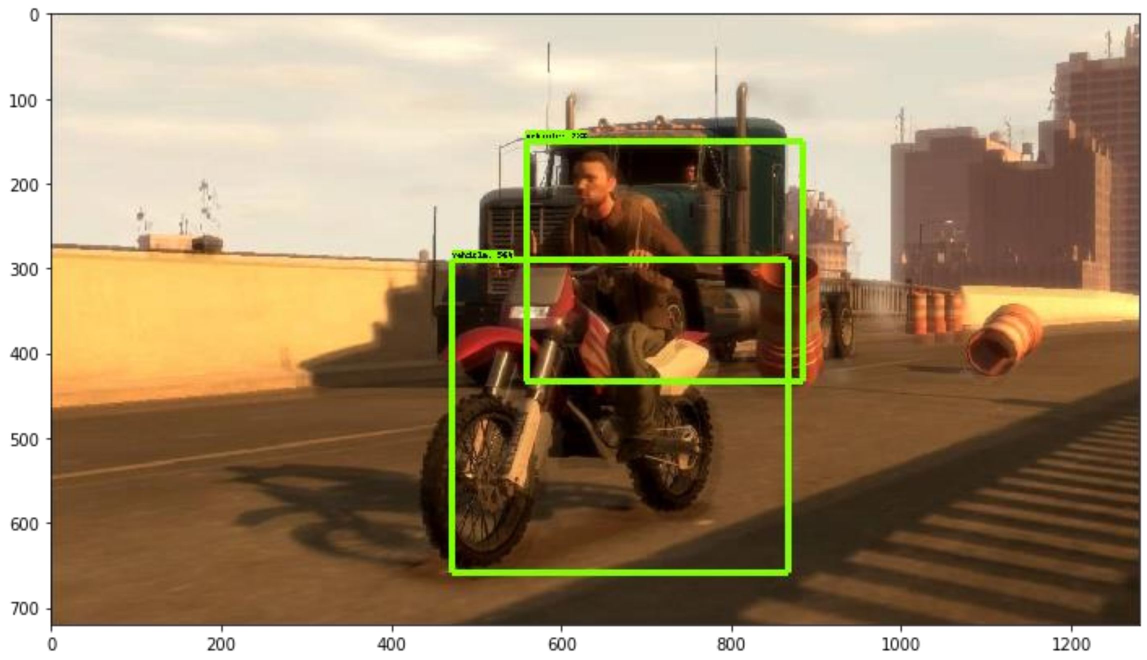


Figure 34 - Test Result

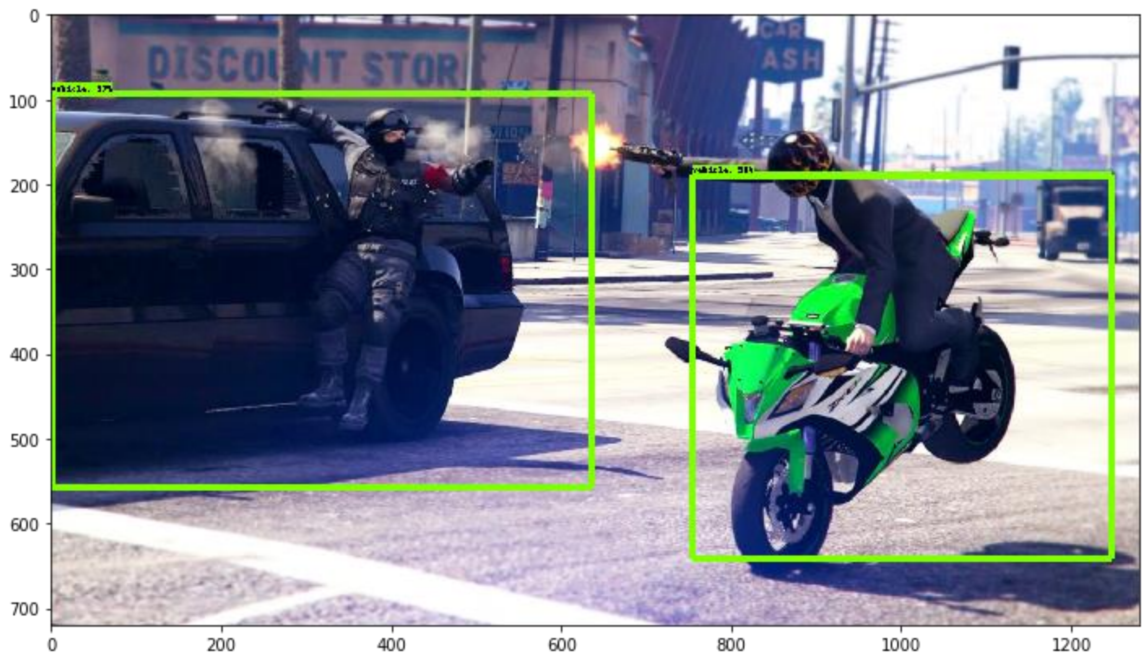


Figure 35 - Test Result

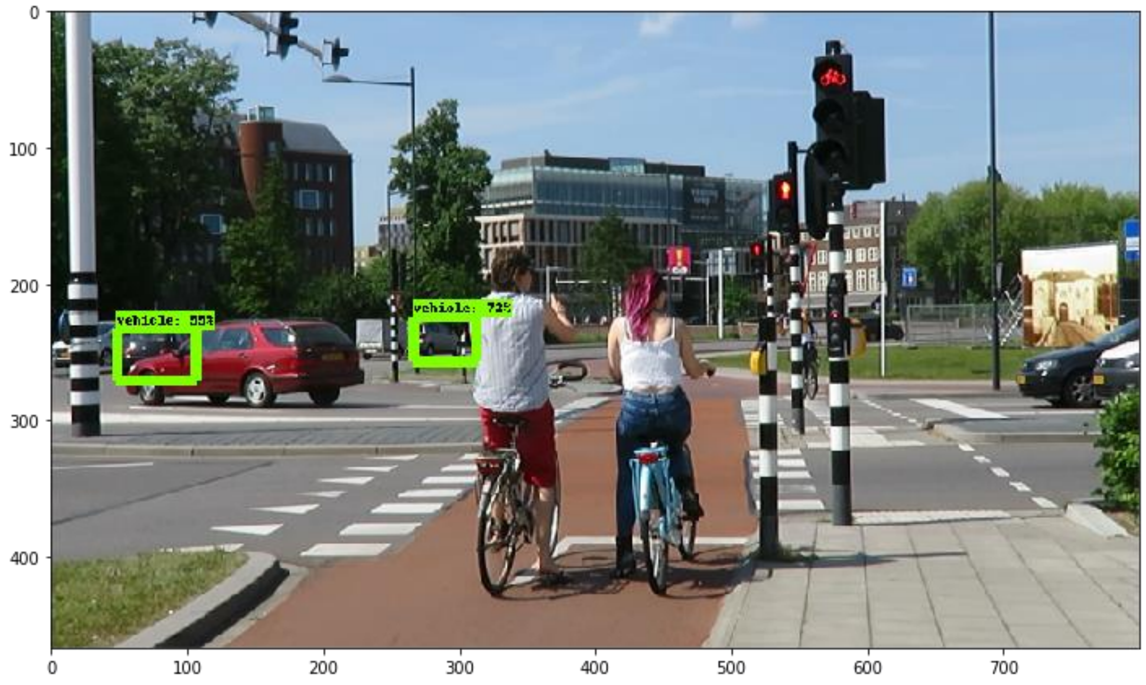


Figure 36 - Test Result

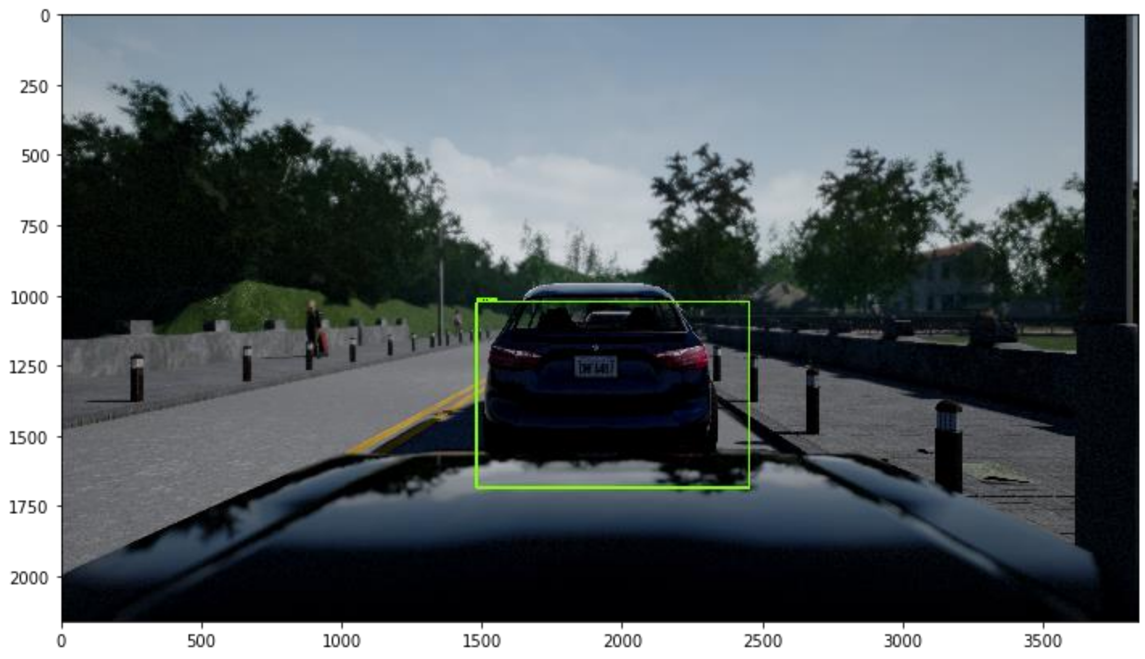


Figure 37 - Test Result

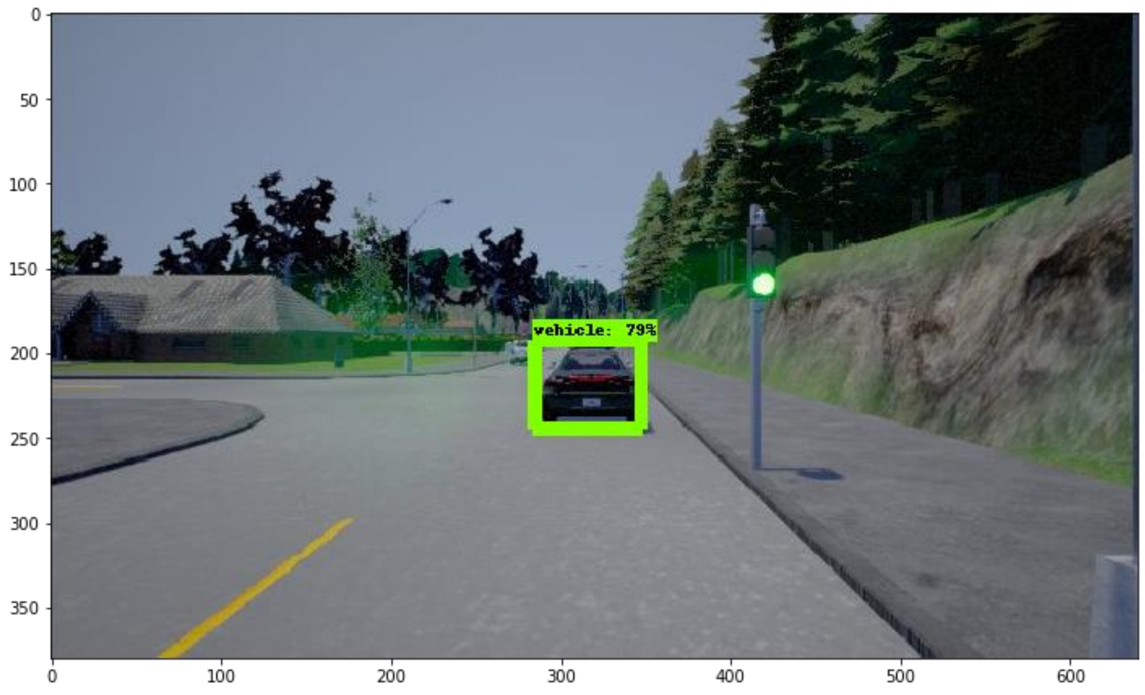


Figure 38 - Test Result

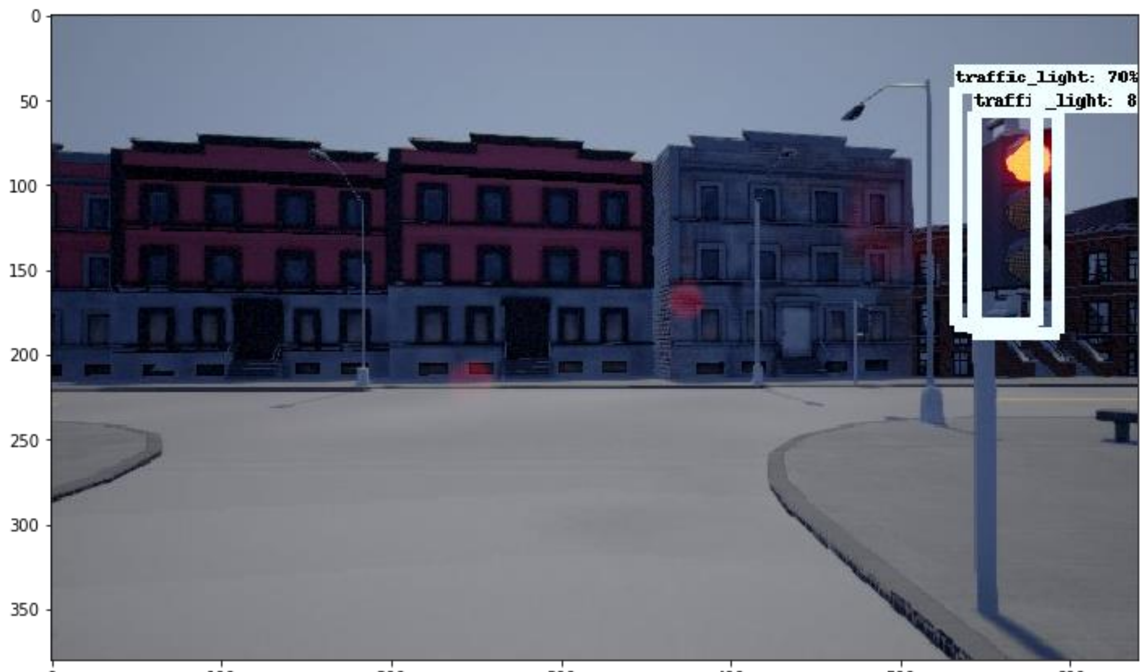


Figure 39 - Test Result

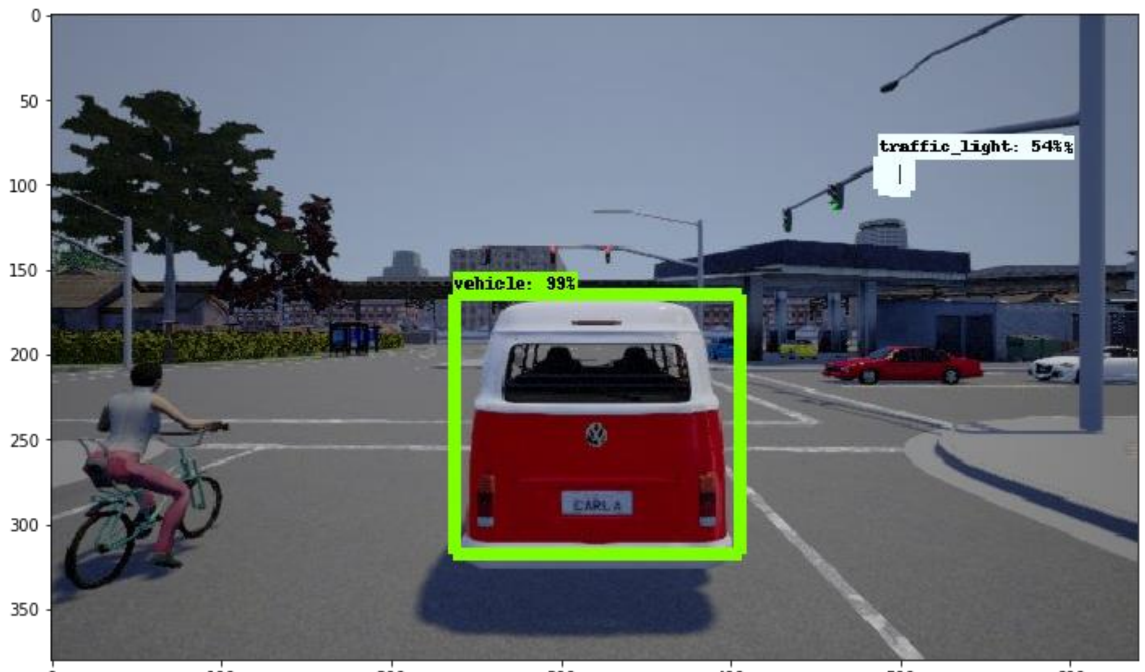


Figure 40 - Test Result

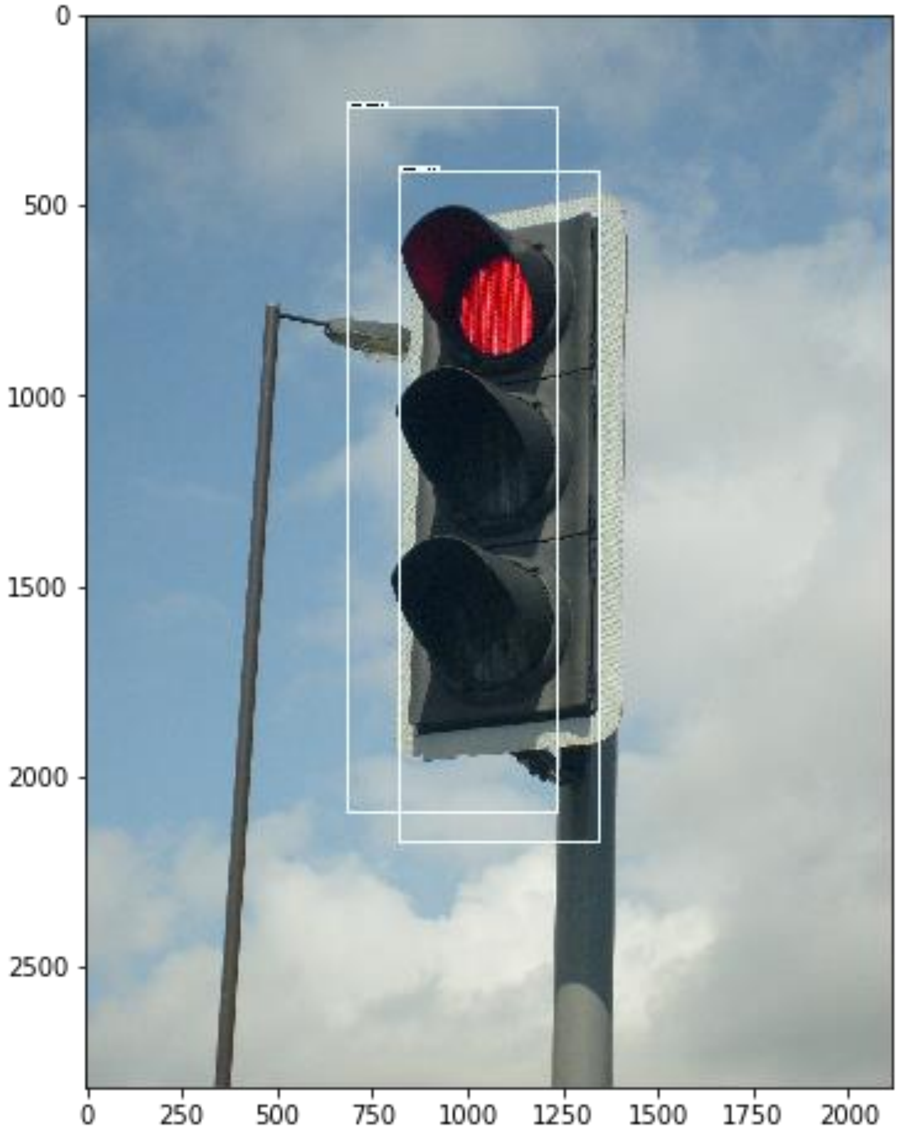


Figure 41 - Test Result

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The data collected from the driving study can further be used to verify and establish value-based dependencies with the personal choices that drivers make. From a large portion of previous work surrounding possibilities in programming moral integrity in robots, it was understood that any semblance of morality or ethics in decisions taken by a machine can only be illustrated by a toy problem at the very basic level. This can be further elaborated to model complex scenarios. The toy problem would ideally consist of a posted speed limit and the degree of adherence to this speed limit by a ‘powerful’ or ‘benevolent’ driver. To program such scenarios, it was proposed that a small-scale driving testbed with multiple robots that emulate autonomous vehicles which can interact among themselves and with their surroundings must be designed. This led to the development of the newer version of CHARTOPOLIS. Further, to improve the scalability of the environment (for use in real world scenarios), it is apparent that cross platform compatibility in neural network models trained across the test bed and the CARLA driving simulator would help in recognizing and grouping different objects into their correct classes from image data. This proof of concept shows that the cross-platform training can be successful with the application of transfer learning. To improve productivity, all these tools should be used in tandem while changing the weights of each neuron in the network (to closely model powerful and benevolent drivers). This is of vital importance as this step can adversely affect the degree to which we can reproduce the composite moral profile identified from participant data.

5.2 Future Work

The future work of this project will consist of the following objectives:

1. Model a city in Unreal Engine that is as similar to the CHARTOPOLIS testbed as possible.
2. Modify the object detection method to increase its accuracy to at least 95%.
3. Toward this end, a new and robust training dataset should be created with at least 30,000 images.
4. All possible benefits of transfer learning should fully be used by extracting models from neural network layers before the output layer and using them as base models for new training.
5. Do a similar study using DenseNet in place of the ResNet architecture for CNN.
6. Conduct imitation learning to closely model the ‘powerful’ and ‘benevolent’ drivers that were identified in Dr. Kathryn Johnson’s studies.
7. Identify crucial object classes pertaining to the toy problem (to be tested in the physical domain, CHARTOPOLIS).

Imitation learning can be a useful tool to edit the weights attributed to each neuron. Weights are assigned randomly when training starts. Gradually they are modified based on the loss function to get desired results. In this case, we can get the weights associated with rash driving or benevolent driving using imitation learning. We can try modifying these to figure out which neuron has a larger association with the moral values that we are trying to program.

REFERENCES

[1] Stevens, William B. “The Automated Highway System Program: A Progress Report.” IFAC Proceedings Volumes 29.1 (1996): 8180-188.

[2] Daisuke Wakabayashi. (2018, March 20). Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam. Retrieved from https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html

[3] Fridman, L., Brown, D. E., Kindelsberger, J., Angell, L., Mehler, B., & Reimer, B. Human Side of Tesla Autopilot: Exploration of Functional Vigilance in Real-World Human-Machine Collaboration. 2019

[4] Fagnant, D. J., & Kockelman, K. (2015). Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. Transportation Research Part A: Policy and Practice, 77, 167-181.

[5] Spring Berman, Nancy Cooke, Mustafa Demir, Ruben Gameros, Sterling Martin, Taylor Reagan, and Rakshith Subramanyam. “CHARTOPOLIS: A Testbed for Driver Interaction with Driverless Cars.” 2018 Southwest Robotics Symposium, Arizona State University, Tempe, AZ, Jan. 25-26, 2018. Oral and poster presentation.

[6] Unreal Engine. (Published: 2004, January 1). Retrieved from https://en.wikipedia.org/wiki/Unreal_Engine

[7] TensorFlow. (Published: 2015, November 9). Retrieved from https://en.wikipedia.org/wiki/TensorFlow

[8] Dai, W., Yang, Q., Xue, G. R., & Yu, Y. (2007, June). Boosting for transfer learning. In Proceedings of the 24th International Conference on Machine Learning (pp. 193-200). ACM.

[9] Kankam, Immanuella. Design of an Immersive Virtual Environment to Investigate How Different Drivers Crash in Trolley-problem Scenarios (2019). Master’s Thesis in Mechanical Engineering, Arizona State University.

[10] Johnson-Roberson, Matthew, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. “Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?” Computer Vision and Pattern Recognition (2016).

[11] tzutalin/labelImg. (Published: 2019, 4). Retrieved from https://github.com/tzutalin/labelImg

[12] Awad, Edmond, Sohan Dsouza, Richard Kim, Jonathan Schulz, Joseph Henrich, Azim Shariff, Jean-François Bonnefon, and Iyad Rahwan. "The Moral Machine Experiment." Nature 563.7729 (2018): 59-64.

[13] Holstein, Tobias, Gordana Dodig Crnkovic, and Patrizio Pelliccione. "Ethical and Social Aspects of Self-Driving Cars." EasyChair Preprints (2018).

[14] Romero, Simon. Wielding Rocks and Knives, Arizonans Attack Self-Driving Cars. (2018, December 31). Retrieved from https://www.nytimes.com/2018/12/31/us/waymo-self-driving-cars-arizona-attacks.html

[15] Baldwin, C.L. and Penaranda, B.N. "Adaptive Training Using an Artificial Neural Network and EEG Metrics for within- and Cross-Task Workload Classification." NeuroImage, vol. 59, no. 1, 2012, pp. 48–56.

[16] Shaoqing Ren, R, et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, 2017, pp. 1137–1149.

[17] Sharma, Pulkit. A Practical Implementation of the Faster R-CNN Algorithm for Object Detection (Part 2 - with Python codes). (2018, November 5). Retrieved from https://www.analyticsvidhya.com/blog/2018/11/implementation-faster-r-cnn-python-object-detection/

[18] Olivas, Emilio. "Handbook of Research on Machine Learning Applications and Trends; Algorithms, Methods and Techniques; 2v." Scitech Book News, vol. 33, no. 4, 2009, pp. Scitech Book News, Vol.33(4).

[19] Brownlee, Jason. How to Reuse Models for Computer Vision with Transfer Learning in Keras. (2019, July 5). Retrieved from https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/

[20] Soekhoe, Deepak, et al. "On the Impact of Data Set Size in Transfer Learning Using Deep Neural Networks." Lecture Notes in Computer Science Advances in Intelligent Data Analysis XV, 2016, pp. 50–60., doi:10.1007/978-3-319-46349-0_5.

[21] Precision and recall. (Published: 2007, November 21). Retrieved from https://en.wikipedia.org/wiki/Precision_and_recall#Definition_.28classification_context.29

[22] Krizhevsky, Alex, et al. "ImageNet Classification with Deep Convolutional Neural Networks." Communications of the ACM, vol. 60, no. 6, 2017, pp. 84–90.

[23] Johnson, K.A., Berman, S., Chiou, E., Pavlic, T.P., Cohen, A.B. (under review). Toward virtuous vehicles: Identifying the moral profile of good drivers as a basis for ethical

decision-making in self-driving cars. Submitted to Basic and Applied Social Psychology, 2019.

[24] Rezatofighi, Hamid, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. "Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression." (2019).

[25] Shaoqing Ren, R., Kaiming He, Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." IEEE Transactions on Pattern Analysis and Machine Intelligence 39.6 (2017): 1137-149.

[26] Kensert, Alexander, Philip J Harrison, and Ola Spjuth. "Transfer Learning with Deep Convolutional Neural Networks for Classifying Cellular Morphological Changes." SLAS Discovery 24.4 (2019): 466-75.

[27] Koopman, Philip, and Michael Wagner. "Challenges in Autonomous Vehicle Testing and Validation." SAE International Journal of Transportation Safety 4.1 (2016): 15-24.

[28] “Trolley Problem.” *Wikipedia*, Wikimedia Foundation, 22 July 2019, en.wikipedia.org/wiki/Trolley_problem.

[29] Rakshith Subramanyam. “CHARTOPOLIS: A Self Driving Car Test Bed.” M.S. thesis, Electrical Engineering, Arizona State University, April 2018.

[30] Dosovitskiy, Alexey, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. "CARLA: An Open Urban Driving Simulator." Proceedings of the 1st Annual Conference on Robot Learning (2017).

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. IEEE Computer Vision and Pattern Recognition (CVPR), 2009.

[32] Lin, Tsung-Yi, 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).

[33] Everingham, Mark, S. Eslami, M. Gool, Ali Williams, Luc Winn, and Christopher Zisserman. "The Pascal Visual Object Classes Challenge: A Retrospective." International Journal of Computer Vision 111.1 (2015): 98-136. Web.

APPENDIX A

SAMPLE CODE FOR IMAGE EVALUATION

Imports

In [0]:

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from distutils.version import StrictVersion
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

# This is needed since the notebook is stored in the object_detection fo
lder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops

if StrictVersion(tf.__version__) < StrictVersion('1.12.0'):
  raise ImportError('Please upgrade your TensorFlow installation to v1.1
2.*.')
```

Env setup

In [0]:

```
# This is needed to display the images.
%matplotlib inline
```

Object detection imports

Here are the imports from the object detection module.

In [0]:

```
from utils import label_map_util

from utils import visualization_utils as vis_util
```

Model preparation

Variables

Any model exported using the `export_inference_graph.py` tool can be loaded here simply by changing `PATH_TO_FROZEN_GRAPH`to point to a new .pb file.

By default we use an "SSD with Mobilenet" model here. See the detection model zoo for a list of other models that can be run out-of-the-box with varying speeds and accuracies.

In [0]:

```
# What model to download.
MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/
'

# Path to frozen detection graph. This is the actual model that is used
for the object detection.
PATH_TO_FROZEN_GRAPH = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')
```

Download Model

In [0]:

```
opener = urllib.request.URLopener()
opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
tar_file = tarfile.open(MODEL_FILE)
for file in tar_file.getmembers():
  file_name = os.path.basename(file.name)
  if 'frozen_inference_graph.pb' in file_name:
    tar_file.extract(file, os.getcwd())
```

Load a (frozen) Tensorflow model into memory.

In [0]:

```
detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')
```

Loading label map

Label maps map indices to category names, so that when our convolution network predicts `5`, we know that this corresponds to `airplane`. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine

In [0]:

```
category_index = label_map_util.create_category_index_from_labelmap(PATH
_TO_LABELS, use_display_name=True)
```

Helper code

In [0]:

```
def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)
```

Detection

```
                                                                    In [0]:
# For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
# If you want to test the code with your images, just add path to the im
ages to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'
.format(i)) for i in range(1, 3) ]

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)
                                                                    In [0]:
def run_inference_for_single_image(image, graph):
  with graph.as_default():
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in op.out
puts}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
          tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
              tensor_name)
      if 'detection_masks' in tensor_dict:
        # The following processing is only for single image
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0]
)
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0]
)
        # Reframe is required to translate mask from box coordinates to
image coordinates and fit the image size.
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], t
f.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_de
tection, -1])
```

```
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num
_detection, -1, -1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_
masks(
            detection_masks, detection_boxes, image.shape[1], image.shap
e[2])
        detection_masks_reframed = tf.cast(
            tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
            detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_te
nsor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: image})

      # all outputs are float32 numpy arrays, so convert types as approp
riate
      output_dict['num_detections'] = int(output_dict['num_detections'][
0])
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.int64)
      output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
      output_dict['detection_scores'] = output_dict['detection_scores'][
0]
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_masks'][
0]
  return output_dict
                                                                  In [0]:
for image_path in TEST_IMAGE_PATHS:
  image = Image.open(image_path)
  # the array based representation of the image will be used later in or
der to prepare the
  # result image with boxes and labels on it.
  image_np = load_image_into_numpy_array(image)
  # Expand dimensions since the model expects images to have shape: [1,
None, None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np_expanded, detect
ion_graph)
  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks'),
      use_normalized_coordinates=True,
      line_thickness=8)
  plt.figure(figsize=IMAGE_SIZE)
  plt.imshow(image_np)
```

APPENDIX B

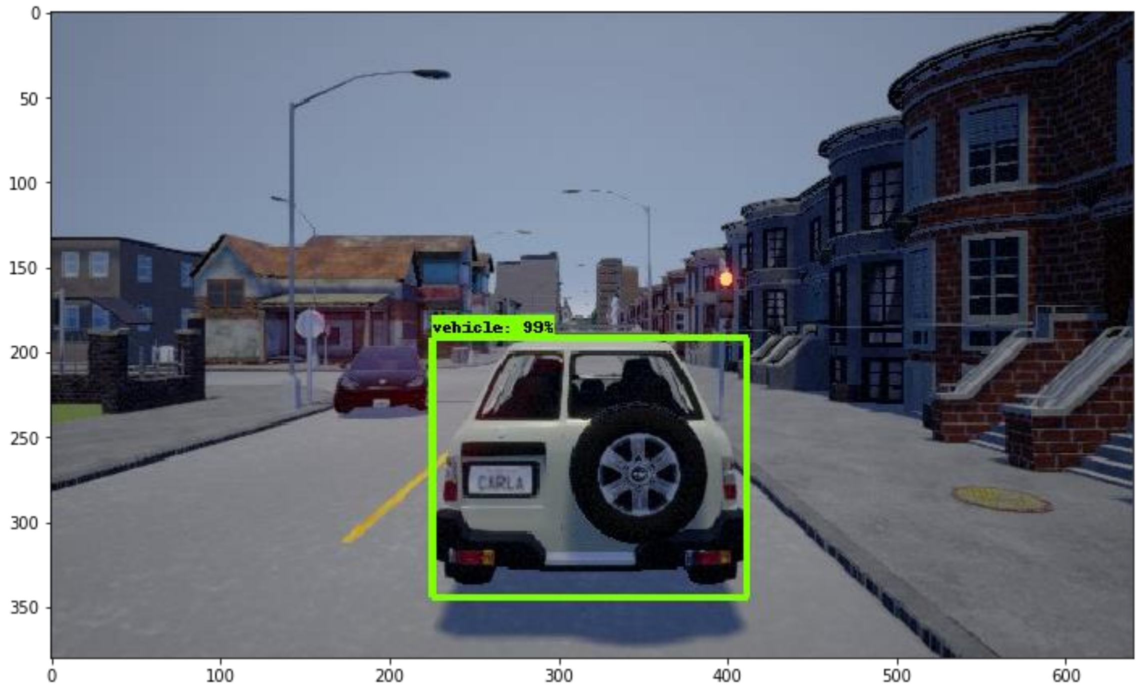
SAMPLE CODE FOR INTERSECTION OVER UNION EVALUATION

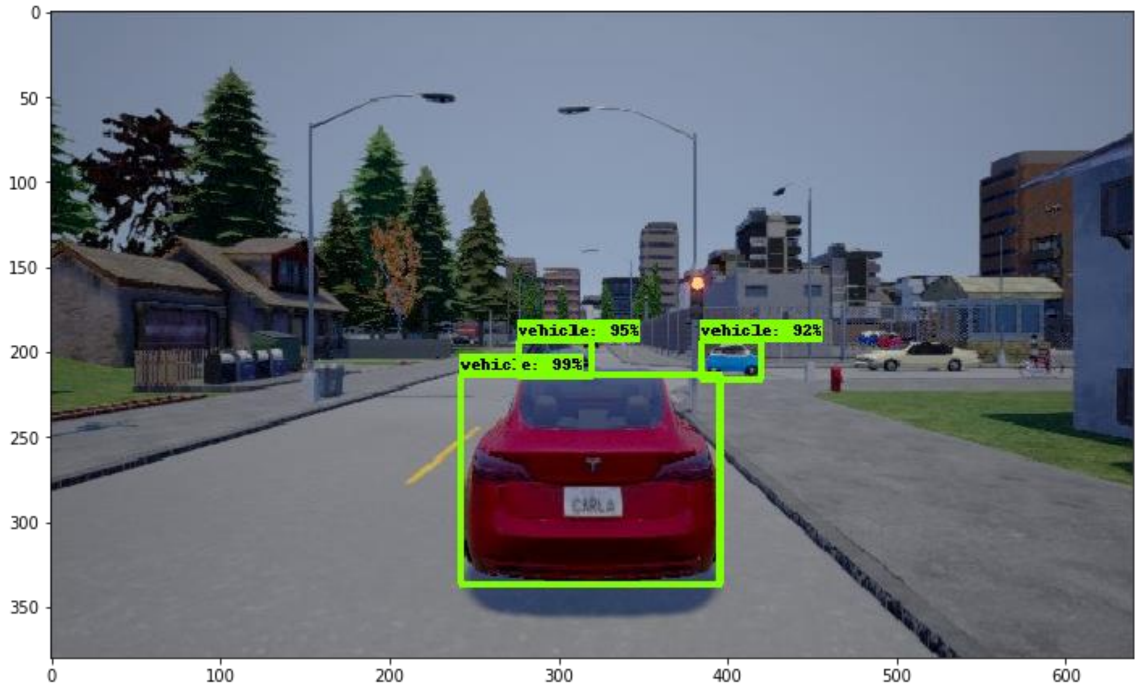
```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


# import the necessary packages
from collections import namedtuple
import numpy as np
import cv2

# define the `Detection` object
Detection = namedtuple("Detection", ["image_path", "gt", "pred"])


# In[2]:


def bb_intersection_over_union(boxA, boxB):
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    # compute the area of intersection rectangle
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)

    # compute the area of both the prediction and ground-truth
    # rectangles
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

    # compute the intersection over union by taking the intersection
    # area and dividing it by the sum of prediction + ground-truth
    # areas - the interesection area
    iou = interArea / float(boxAArea + boxBArea - interArea)

    # return the intersection over union value
    return iou


# In[3]:
```

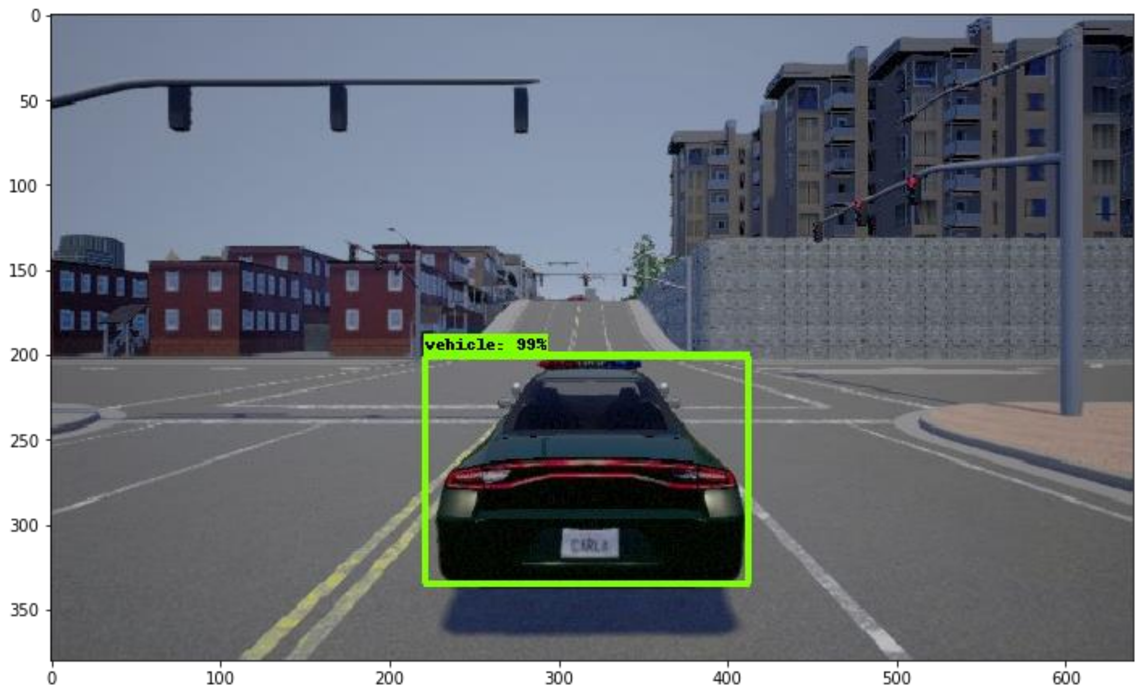
```
# define the list of example detections
examples = [
    Detection("image1.jpg", [529, 40, 581, 168], [530, 44, 576, 170]),
    Detection("image2.jpg", [225, 184, 414, 337], [227, 186, 416, 339]),
    Detection("image3.jpg", [243, 207, 397, 336], [245, 209, 394, 333]),
    Detection("image4.jpg", [227, 203, 411, 338], [225, 200, 415, 342]),
    Detection("image5.jpg", [455, 205, 640, 305], [456, 206, 644, 310])]


# In[4]:


# loop over the example detections
for detection in examples:
    # load the image
    image = cv2.imread(detection.image_path)

    # draw the ground-truth bounding box along with the predicted
    # bounding box
    cv2.rectangle(image, tuple(detection.gt[:2]),
        tuple(detection.gt[2:]), (0, 255, 0), 2)
    cv2.rectangle(image, tuple(detection.pred[:2]),
        tuple(detection.pred[2:]), (0, 0, 255), 2)

    # compute the intersection over union and display it
    iou = bb_intersection_over_union(detection.gt, detection.pred)
    cv2.putText(image, "IoU: {:.4f}".format(iou), (10, 30),
        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
    print("{}: {:.4f}".format(detection.image_path, iou))

    # show the output image
    cv2.imshow("Image", image)
    cv2.waitKey(0)
```

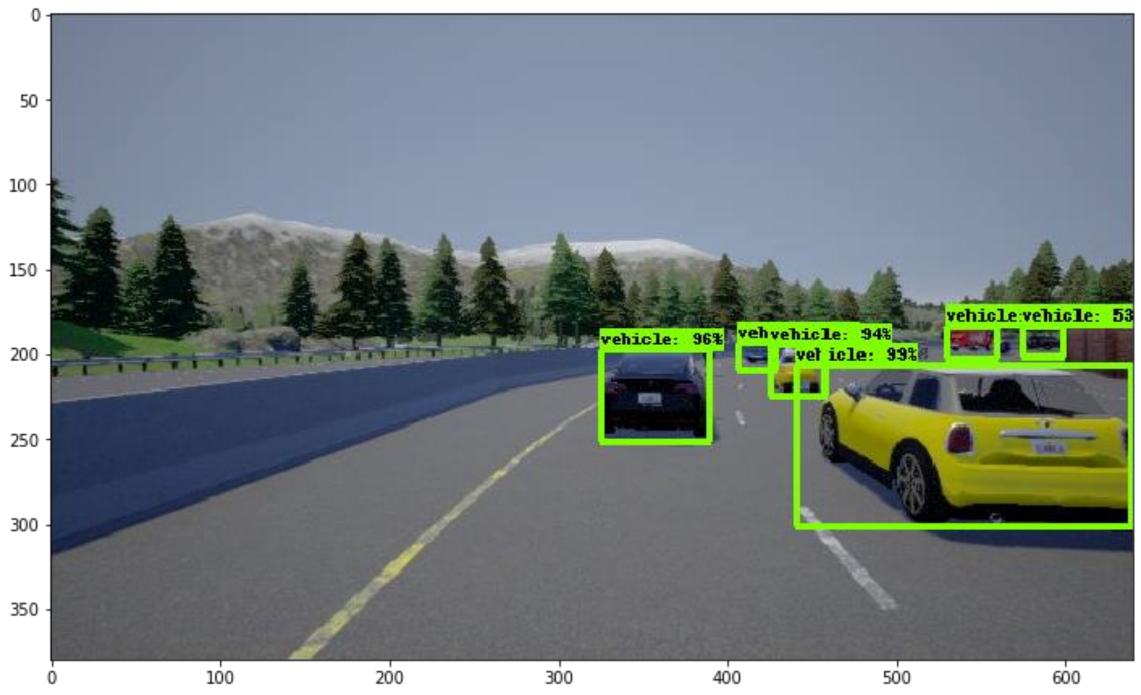
APPENDIX C

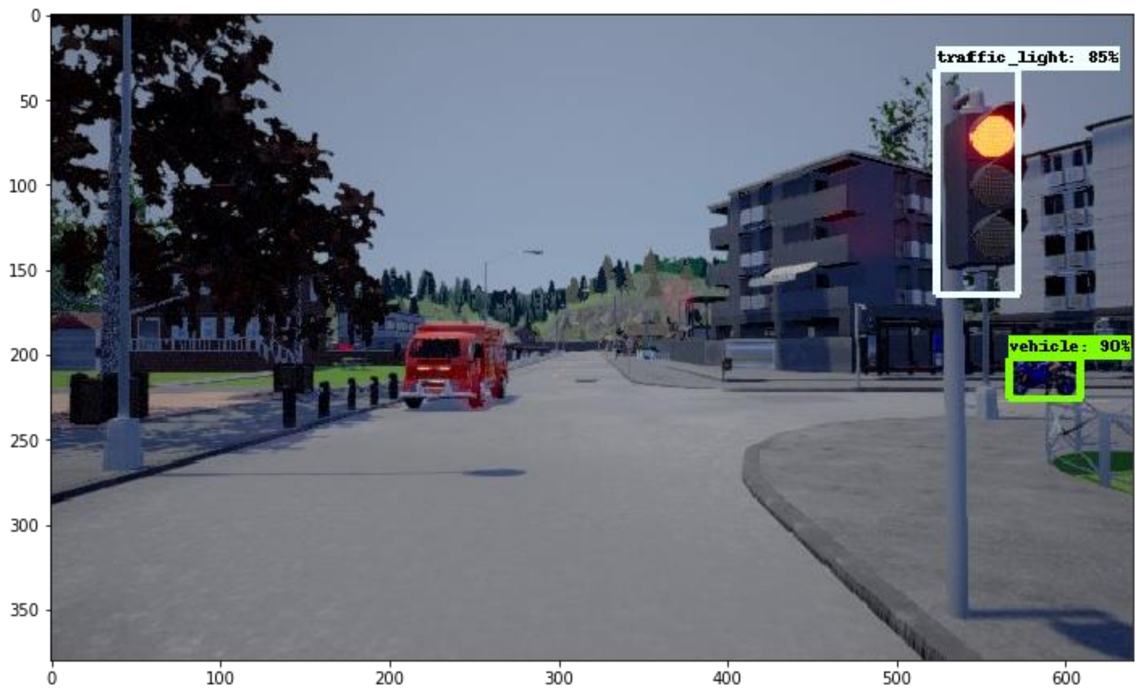
OBJECT DETECTION RESULTS FOR IOU TEST DATASET

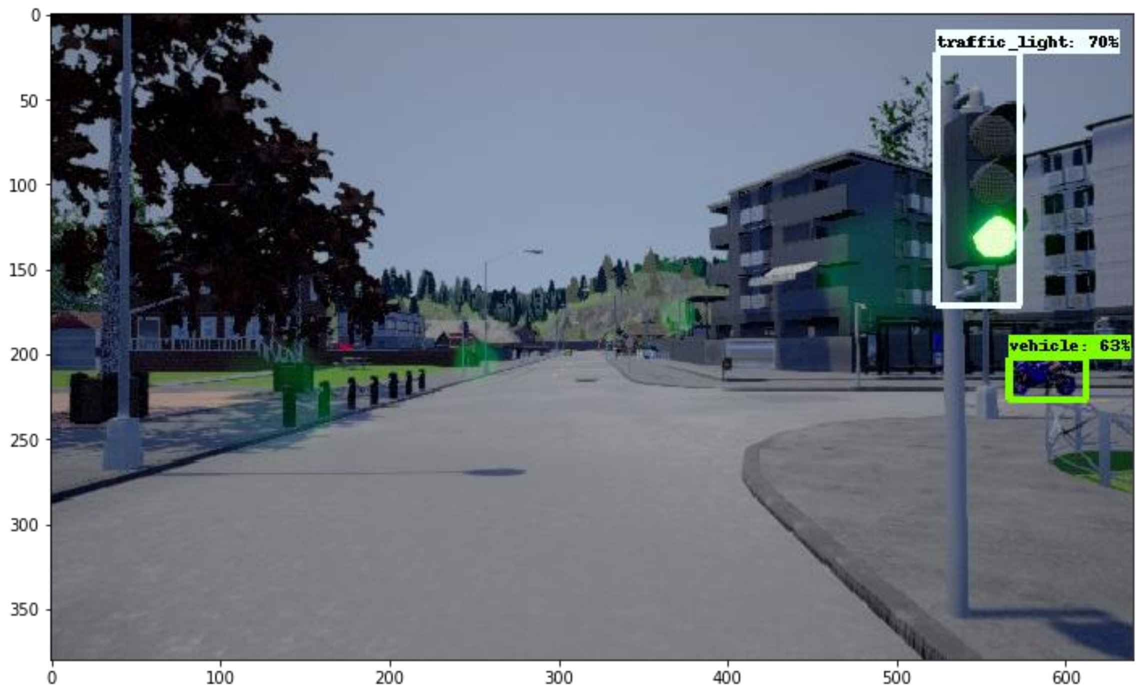

vehicle: 99%

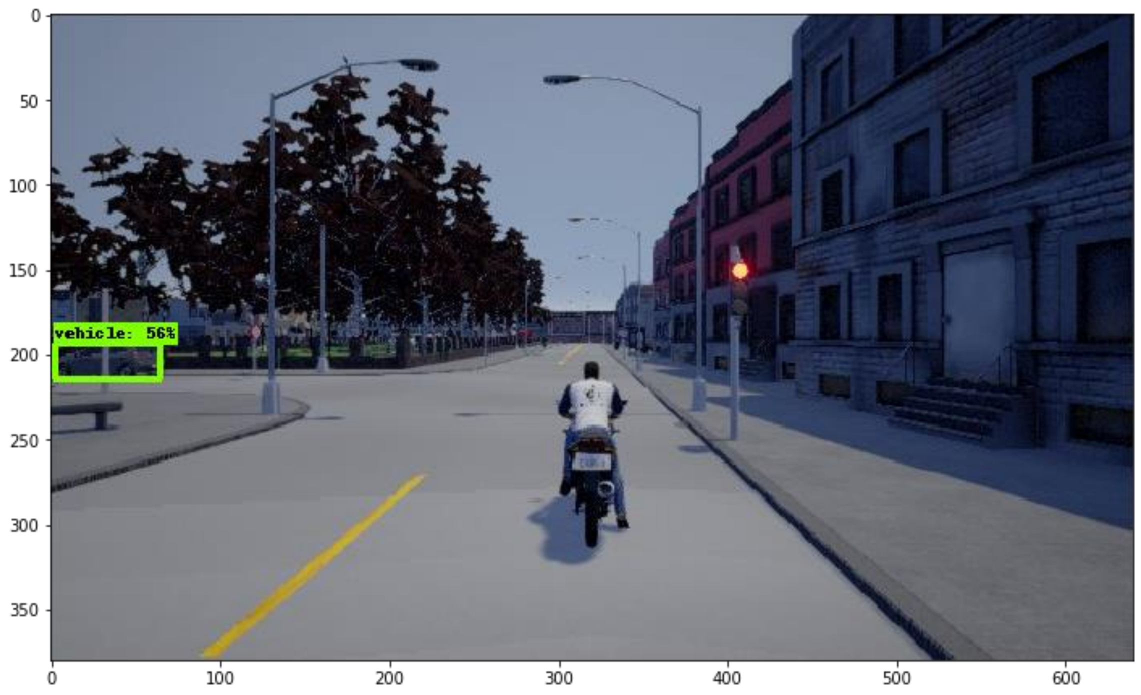

vehicle: 95%
vehicle: 92%
vehic e: 99%

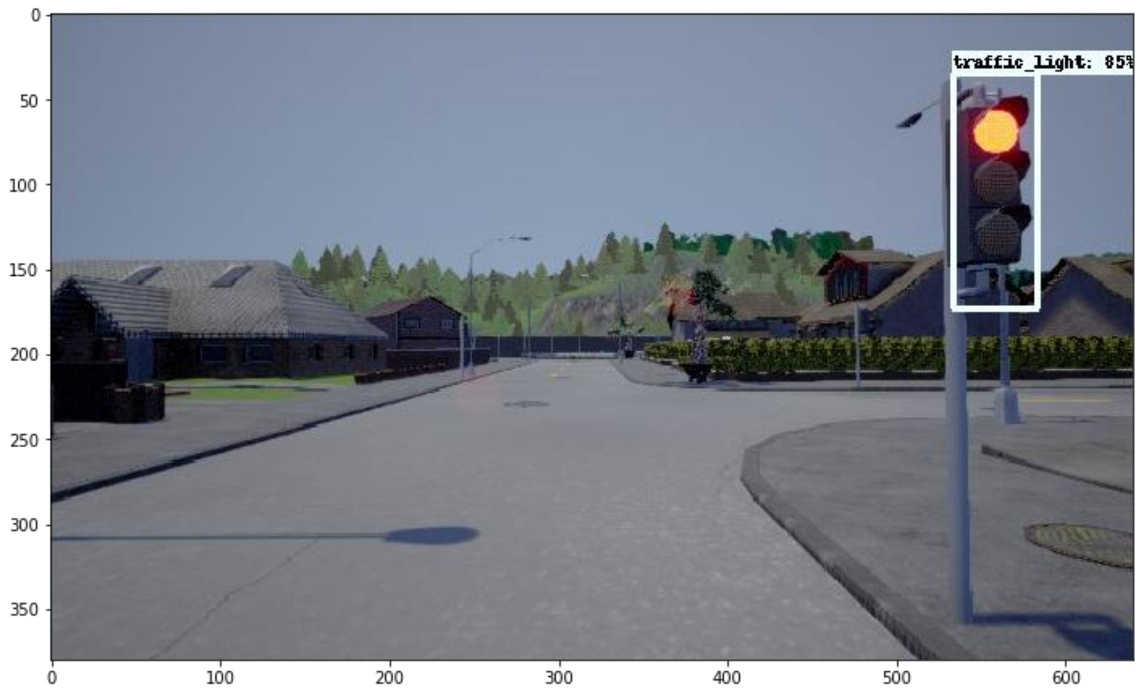

vehicle: 99%


vehicle: 96%
veh
vehicle: 94%
vehicle: 99%
vehicle
vehicle: 53


traffic_light: 85%
vehicle: 90%
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600


traffic_light: 70%
vehicle: 63%
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600


vehicle: 56%


traffic_light: 85%