Implementing YOLOv3 Object Detection on the Berkeley Deep Drive Dataset

Final Report

Jacob Ieyoub
Springboard
Houston, TX
jieyoub@gmail.com

***Abstract* - The future of transportation is self-driving cars. In the same way that cars replaced horses as the common mode of transportation 100 years ago, self-driving cars will take over the inefficient ways cars are used today. Society as we know it will change forever once autonomous vehicles are commercialized. An important aspect of self-driving cars will be real-time computer vision software that will be able to accurately perceive what is on the road so that the car can make safe decisions while driving. The goal of this project is to implement YOLO v3 for real-time object detection using the Berkeley Deep Drive dataset. The model will aim to detect ten classes of common objects encountered while driving: bikes, buses, cars, motorcycles, pedestrians, motorcycle/bike riders, traffic lights, traffic signs, trains and trucks. Google Colab will be utilized so that the training can be done using a GPU in the cloud.**

1. Introduction - The inspiration for this project is the book *Autonomy: The Quest to Build the Driverless Car - and How It Will Reshape Our World* by Lawrence D. Burns. The inevitability and necessity of commercial self-driving cars did not really hit me until I found this book. Burns does a great job of illustrating the impact that self-driving cars will have on our society. He considers self-driving car development to be the "space race of our time." Discussing what all will be affected by this shift in our paradigm is pretty eye-opening. To begin, self-driving cars will improve the safety of transportation dramatically. Burns estimates that 90% of all automobile deaths can be stopped from self-driving cars. There are about 1.3 million global deaths per year caused by automobile accidents. On top of that, it is commonly accepted that the autonomous fleets of the future will use either electric battery, hydrogen fuel cell or other alternative propulsion systems. This means the environmental effects from cars will be dramatically reduced. The demand for oil will greatly decline which will vastly change the geopolitical landscape. There will no longer be drunk driving and highway patrolling by law enforcement will be largely cut out. The resources allocated by law enforcement for transportation can be allocated to fight more important crime. There are little things that would be affected as well. Transportation as a service would be the modus operandi of these vehicles and the general idea is that a company would deploy a fleet for customers to subscribe to on a monthly, annual or per use basis. Therefore, no one would be buying cars anymore. Car

insurance, car dealers and even garages would be a thing of the past. Houses will be built completely differently. It is truly amazing how much society will change from self-driving cars. Not to mention the massive economic impact this shift will have. Burns estimates that the cost right now to drive a car is roughly $1.50 per mile if you also account for the time you lose while driving. From there he estimates that with self-driving cars the cost will decrease to $0.20 per mile for the consumer. It will not be technology for the sake of technology but will absolutely change our lives for the better. This all made me want to explore the deep learning side of a self-driving car and object detection is a good place to start.

2. You Only Look Once- YOLO is a convolutional neural network (CNN) built for real-time object detection. Convolutional neural networks are great for pattern recognition within images. They are able to filter images in a way that recognizes basic patterns to start, such as edges, circles, etc., and eventually complex patterns as the network gets deeper, say an eye, ear or nose. This is accomplished by convolving images.

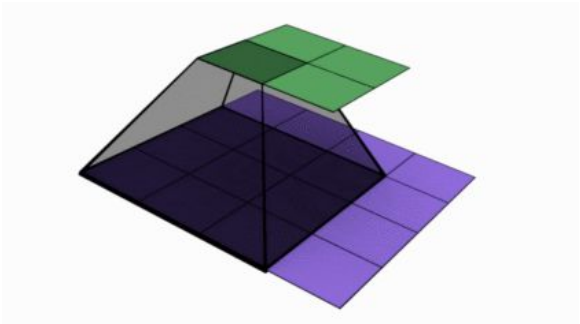

Figure 1. Convolution

A node within a hidden layer has a filter which is a matrix of random numbers and, for example in a 3x3 filter, the dot product of the filter and every 3x3 matrix within the image is calculated and the results of these dot products form another matrix. The patterns begin to be recognized and as the network deepens, objects are detected and bounding boxes predicted.

YOLO v3 is quite fast but sacrifices a bit of accuracy for speed. When speed is an important factor, for example in real-time detection, it is the go-to algorithm for object detection. Object detection is the process of not only recognizing and classifying items in an image but also showing where in the image that object is. The algorithm accomplishes this by predicting where bounding boxes are located and then classifying the object within that bounding box.

The structure of YOLO v3 contains fifty-three convolutional layers and is aptly named Darknet-53. Darknet-53 is fully convolutional. Each layer is followed with batch normalization and a “Leaky ReLu” activation. Batch normalization is a technique that standardizes the inputs to a layer. This technique can reduce training time and the amount of epochs that are needed for training. A Leaky ReLu is an activation similar to a ReLu (rectified linear unit) activation except that the output will not be zero whenever $x < 0$ but will have a small negative slope. This is designed to solve the issue of a “dying ReLu.” This is the issue of a neuron “dying” when the weights are configured in a way so that neuron is never activated. In other words, the output would always be zero. The Darknet-53 architecture can be seen in ***Table 1***.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. Darknet-53

YOLO v3 begins by making bounding box predictions using logistic regression. Four coordinates are predicted. These include the x and y center coordinates, the width and the height of the bounding box. An objectness score is predicted with a value of 1 if a bounding box that is predicted overlaps with the ground truth object more than any of the other predicted bounding boxes. Something of note that should be pointed out is that for class prediction in YOLO v3, softmax activation is not used. This is because softmax assumes only one class for each box. A popular dataset for implementing YOLO is the Open Images Dataset which has overlapping labels. For example "animal" and "chicken" could be assigned to the same box.

The fifty-three convolutional layers of Darknet-53 make it much more powerful than its predecessor Darknet-19. It is also much faster than other popular networks such as ResNet-101 or ResNet-152 while not sacrificing all that much accuracy. Many metrics can be looked at to compare object detection networks. Most are differing thresholds of AP (average precision). Table 2 (on the next page) does a great job of showing these metrics for different networks. AP_{50} is the AP with the threshold at 0.5 and AP_{75} is for a threshold of 0.75. AP_S, AP_M and AP_L are for small, medium and large objects, respectively. YOLOv3 scores very high in small object detection which it failed to do in past versions. In addition, Figure 2 (also on the next page) does a great job of visualizing the speed at which YOLO v3 performs and still keeps its precision. As you can see it is far and why the fastest network for how accurate it is.

These are some general insights into how CNNs and, specifically, YOLO work. It is a great network and an effective way to train a custom dataset for object detection. The next phase in the project will be implementing YOLO v3 to train the nine objects in Berkeley Deep Drive dataset by using Google Colab. We will now dive into the Berkeley Deep Drive dataset set and the analysis of the annotated images. Then, the process of training the neural network will be detailed, along with evaluating the performance, lessons learned and future work.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Table 2. AP values at different thresholds.

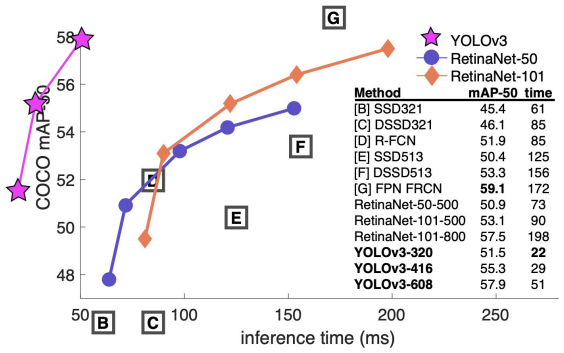

Figure 2. mAP-50 vs Inference Time.

3. Exploratory Data Analysis - The Berkeley Deep Drive dataset was analyzed using the annotation files for the bounding boxes. I determined the distribution of varying image characteristics. This included class, weather distribution, scene distribution and time of data distribution. There is much imbalance in the class distribution. The car class has the largest sample size at 713,211 and the train class has the smallest sample size at 136. Sample size has a direct relation to the AP of each class. More on that later. In the other distributions, clear, city street and daytime have the largest instances in weather, scene and time of day, respectively. These distributions can be seen in Figure 3. Along with class and image characteristic distributions, bounding box size and bounding box center distributions were observed. The Berkeley Deep Drive dataset has a lot of small objects annotated. This is what can make it so challenging for object detectors. The density plot for bounding box sizes by class can be seen in Figure 4. The x-axis is the square root of the bounding box area and the y-axis is the probability that the bounding box will be that size. As you can see the majority of bounding boxes are smaller, especially in the traffic light and traffic sign classes.

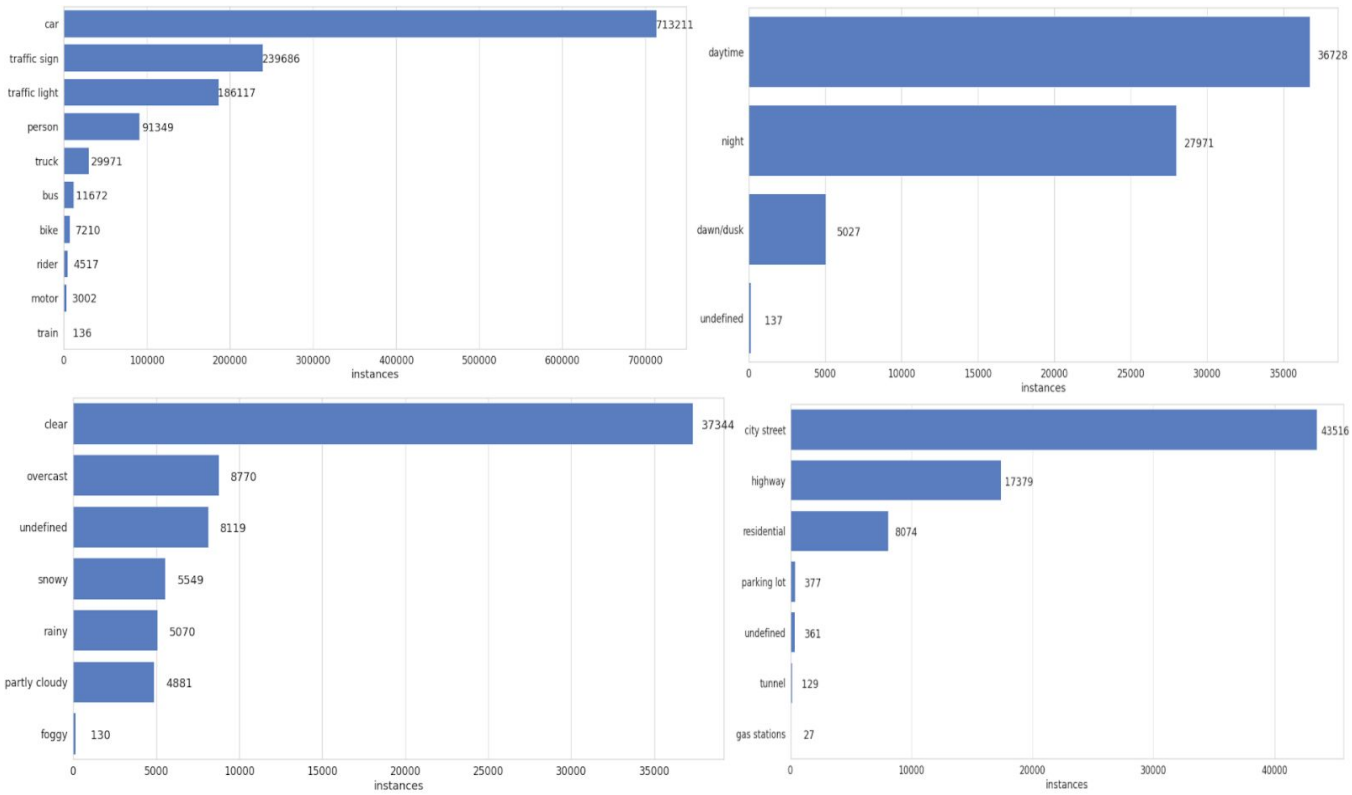

Figure 3. Distributions of class, time of day, weather and scene.

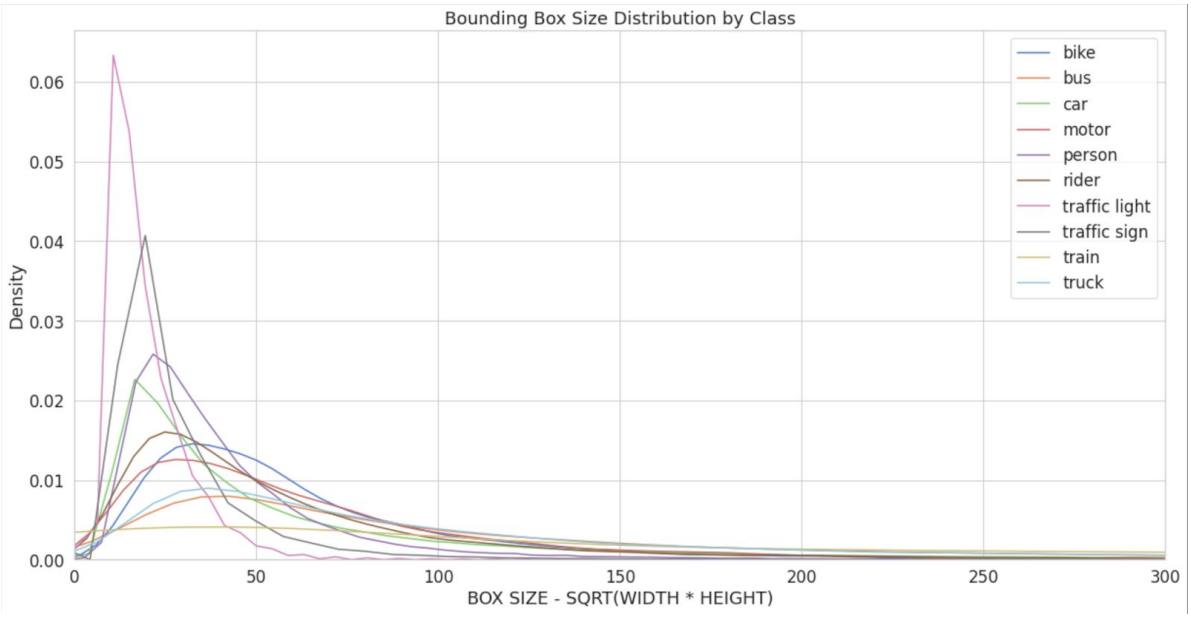

Figure 4. Bounding Box Size Distribution by class.

We also looked at the distribution of bounding box centers within any given image from the dataset. It is an easy way to visualize where most of the objects occur within the dataset. This can be seen in ***Figure 5***. It appears that most objects occur in and around the middle of the image, this is not surprising as the images are from street-level view through the windshield of a vehicle. The axes represent pixels and as our images are 1280 x 720, the x-axis is 1280 pixels and y-axis is 720 pixels. These visualizations give some great insights into how the imageset data looks. These insights aid in the interpretation of our model performance which will be discussed in subsequent sections.

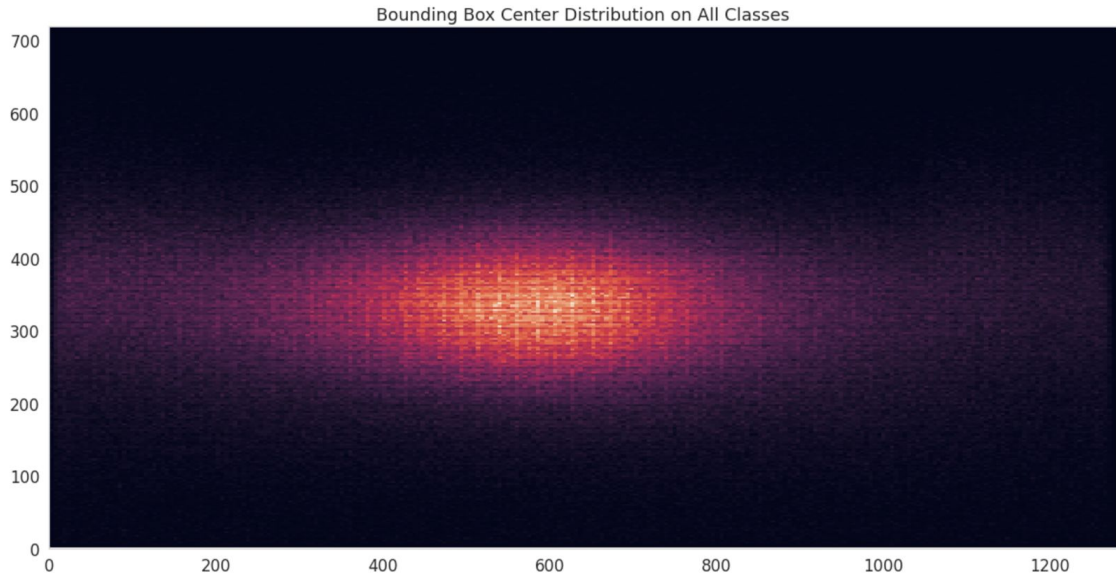

Figure 5. Bounding Box Center Distribution.

4. Training - Training of our network was implemented using the darknet framework. To do this at a reasonable speed, Google Colab was used so that a GPU could be used in the cloud. There were some stipulations to this and these will be discussed a little later on. The files required for training are as follows:

1. image files (<img_name>.jpeg)
2. image annotations (<img_name>.txt)
3. data.data file
4. classes file (classes.names)
5. configuration file (<cfg_name>.cfg)
6. paths to training images (train.txt)
7. paths to test images (test.txt)
8. pretrained weights file (.weights)

The image files are all the images you are using for training in jpeg format. The image annotations are the Berkeley Deep Drive annotated bounding box for each image. The format needed for YOLOv3 is (<class>, <x_center>, <y_center>, <width>, <height>). These coordinates are determined from the json files that contain the image bounding box annotations. However, they come in the (x_min, y_min, x_max, y_max) format and they are also in pixels, with the range 0-1280 for x_min, x_max and 0-720 for y_min, y_max. The coordinates for YOLOv3 training are all relative to the image width or height, so they are between 0 and 1.

The data.data file contains the following:

1. number of classes
2. path to train.txt
3. path to test.txt
4. path to classes.names
5. desired path to save weights during training

The classes.names file consists of all the classes and the number assigned to that class for training. The configuration file contains the YOLOv3 network that is adjusted to your custom dataset. We will need to change a few parameters of our configuration file for our dataset.

1. batch = 64
2. max_batches = 20,000
   (2000 * num_classes)
3. steps =16,000, 18,000
   (80%, 90% of max_batches)
4. *in the [yolo] layers*
   classes = 10
   (num_classes)
5. *in the [convolutional] layers right before the [yolo] layers*
   filters = 45
   ((classes + 5) * 3)

The train.txt and test.txt files contain the paths to all the images for training and testing, respectively. And lastly, the pretrained weights file is the weights you want to begin training with. If you are just beginning, it is wise to use the darknet beginning weights. If you have already trained the network up to a certain point and want to continue training you would start with the most recent weights you have.

One of the stipulations of training a neural network on Google Colab is the environment runtime can expire during training if you leave your computer. This can get frustrating if you are trying to graph the loss during training, which is why I do not have a graph showing how the loss decreased over iterations.

The model was trained over 8000 iterations. The loss was around 6 and from there it was time to evaluate its performance on each individual class and the overall mAP of all ten classes.

5. Evaluation - There are two metrics used in unison for evaluation of object detectors. These are Intersection Over Union (IoU) and Average Precision (AP). These are used

together to determine the mean average precision (mAP) of an object detector. The mAP is the mean of all the individual class APs and it is usually calculated at a specified IoU. For example mAP_{50} is the mAP at IoU = 0.5 or 50%. The evaluation is run on the test set of images and the results can be seen in ***Table 3***. The results have a direct relation to sample size, especially for the car, traffic sign, bike, motor, rider and train classes. Traffic light, the third largest sample size by class in the training set, does not have the third highest AP amongst classes. This could be for a number of reasons.

class	AP
car	58.96
traffic sign	50.01
truck	47.27
bus	43.60
traffic light	36.12
person	31.73
bike	25.27
rider	23.94
motor	15.04
train	0.00
mAP	33.19

Table 3. AP results.

If you look at ***Figure 4***, traffic light has the highest density of smaller images relative to the other classes. This could be a factor for its lower AP to bus and truck, even though they have a smaller sample size. Another factor could be that traffic lights are not usually in the immediate line of sight that the majority of object centers are as seen in ***Figure 5***. Since They are usially higher up within the image and smaller, this could affect the performance of our model when detecting traffic lights. The same could be said for the person class. They are smaller objects within the image and therefore can be harder to detect. It is no surprise that the train class has an AP = 0.00. There were only 136 instances of a train in the dataset. This is super low, especially when compared to the car class which has 713, 211 instances. For the most part, our network performs well. This was pretty much out of the box implementation and to get an mAP of 33.19% is quite strong.

6. Conclusion - To visualize the performance of our object detector, I ran the detection on a homemade video while driving my car. It works well and it is rewarding to see the model in action detecting cars, traffic signs, etc. YOLOv3 is a great starting point for an aspiring computer vision engineer. It is effective, fast and practical. With Google Colab, it is possible to implement without purchasing a GPU or paying for a cloud computer service. This provides the ability to experiment and re-run training cheaply. Some takeaways from our model performance show that sample size and object size is a big factor in detecting objects accurately.

7. Future Work - Berkeley Deep Drive is a challenging dataset. To be more accurate, many parameters would need to be optimized and the model would need to be trained longer. In the video, some inconsistencies can be seen. Whether it be an object on the side of the road classified as a traffic sign or a car that is detected in some frames but not others. Object tracking would

be a useful endeavor to implement into this model. This would improve consistent detection of an object throughout the video. It would also be beneficial to add more images with trains to improve the AP on the train class. This could provide huge gains in mAP cheaply. Parameters that could be experimented with include the batch size during training and also the image input size. We used a batch size of 64, but maybe a batch size of 32 or 128 could be better. We used the default 416 x 416, and it is possible a different size could improve performance. Learning rate is another parameter that can be optimized in our model. The default is 0.001. There are also newer versions of YOLO. It would be beneficial to explore these models and see how they are better than YOLOv3.

Object detection is just the tip of the iceberg when it comes to computer vision for self-driving car software. You need lane detection and might need 3D object detection that would assign three dimensional bounding boxes to an object or semantic segmentation that assigns each pixel to a class. This does not even begin to go into the AI that would make the decisions after perceiving what around the car. The complexities of self-driving cars is quite mind-boggling and it is amazing that we are on the cusp of using this technology on a daily basis. It will change the world as we know it and for the absolute betterment of society. Buckle up and enjoy the ride.

The github repository for this project can be found here.

8. References

1. F. Yu, et. al., *Berkeley Deep Drive,* Accessed on: June 15, 2020. [online]. Available: <https://bdd-data.berkeley.edu/>
2. L. Burns, *Autonomy : The Quest to Build the Driverless Car - And How It will Reshape Our World.* New York, NY : HarperCollins. 2018.
3. J. Redmon, A. Farhadi, *YOLOv3 : An Incremental Improvement*, University of Washington, April 8, 2018. Accessed on: May 27, 2020. [online]. Available: <https://pjreddie.com/media/files/papers/YOLOv3.pdf>
4. J. Redmon, A. Farhadi, *YOLO:Real-Time Object Detection.* 2018. Accessed on: May 27, 2020. [online]. Available: <https://pjreddie.com/darknet/yolo/>
5. The AI Guy, *YOLOv3 in the Cloud,* Jan. 7, 2020. Accessed on: June 15, 2020. [online]. Available: <https://www.youtube.com/watch?v=10joRJt39Ns&t=2175s>
6. Darknet Repository. Accessed on: June 15, 2020. Available: <https://github.com/AlexeyAB/darknet>
7. R. Kalliomaki, *Real-time object detection for autonomous vehicles using deep learning,* Uppsala Universitet, June 2019. Accessed on: July 20, 2020. [online]. Available: <https://uu.diva-portal.org/smash/get/diva2:1356309/FULLTEXT01.pdf>