Real-time Detection & Tracking of Mobile Objects on Roads For Autonomous Vehicles using Deep Learning

A Project Report
Presented to
The Faculty of the Department of Applied Data Science
San Jose State University
In Partial Fulfillment Of the Requirements for the Degree
Master of Science in Data Analytics

By

Angie Gao
Darshit Jagetiya
Max Ng
Haoran Shi

December 10, 2021

APPROVED FOR DEPARTMENT OF APPLIED DATA SCIENCE

Dr. Jerry Gao, Project Advisor

Dr. Lee C. Chang, Department Chair

ABSTRACT

Real-time Detection & Tracking of Mobile Objects on Roads for Autonomous Vehicles using Deep Learning

By
Angie Gao, Darshit Jagetiya, Max Ng, and Haoran Shi

I. Purposes

Nowadays, traffic accidents have been the eighth leading cause of death for children and teenagers who are 5-29 years old. Every day, around 3,700 people lost their lives in road crashes around the world. Every 24 seconds, someone dies on the road.

According to the National Highway Traffic Safety Administration (NHTSA), distracted driving is the leading cause of car accidents and truck accidents in the U.S. Distracted driving is so hard to prevent because human's attention is very easy to be distracted. Talking to passengers, paying attention to a child or pet, or even thinking about problems can cause a driver to not pay attention to the road marks and traffic signs. Therefore, in this project, we have developed our model to detect, recognize and classify traffic signs, road markings, and other mobile objects on roads using deep learning algorithms. We hope our model can guide drivers to stay safe during their trips.

Besides, traffic signs and road markings detection and classification in real-time is fundamental to driver assistance systems and self-driving technology. A good driver assistance system requires velocity (real-time), veracity (accuracy) and variety (different kinds of traffic signs and marks). These 3 Vs are also the target of our model. Some existing research focuses only on detecting traffic signs, but our goal is more than that. Although ignored by many developers, road markings are also a vital part of our road infrastructure. Transverse markings, pavements and guiding arrows also provide important information to drivers. Our model focuses on improving the existing research by adding more objects for detection for a self driving vehicle. The ultimate purpose of our model is to provide a comprehensive solution to drivers in real-time.

II. Tasks

The scope of this project is to develop a deep learning model that is able to take the real-time video input with the eye of an autonomous vehicle which can detect all the pavement marks, traffic signs, vehicles, and other pedestrians to classify them and calculate the distance

between the vehicle and these objects to suggest the step the vehicle should take. Most of the similar research work has concentrated on detecting traffic lights, traffic signs, or lanes but road marks play a significant role for an autonomous vehicle. The previous research work also does not consider calculating the distances of the detected traffic signs which is implemented in the model to make decisions for the system of self-driving vehicles based on the type of the sign classified.

To develop such a model, sufficient data was required for training to classify the large number of objects being considered for classification. Labeled traffic sign datasets are widely available from past research. For this project, one of the key aims was to detect and classify all the pavement marks whose labeled data is not available. To collect all the types of road marks, videos were taken to get images at different distances. These images were labeled manually for classification which were utilized for training the model as the past research mostly consisted of labeled data for traffic lights and signs only.

The model also detects bikers, trucks, and 90 other classes of COCO dataset in addition to roadmarks and traffic signs by ensembling methods. The modeling part was divided into multiple sections. The key step in modeling is to train a deep learning model to detect and classify the traffic signs and road marks. Recently developed Convolutional Neural Network method known as You Only Look Once v4, and EfficientDet1 have shown promising results for accuracy, speed requirement for real time detection, and issues with small object detection problems which will be used for detection and classification (Rajendran et al., 2019). Models like Faster RCNN and CenterNet are also state of the art object detection models. For the detection of traffic lights, pedestrians, traffic signs, and vehicles, numerous models are already trained due to past research work available for which transfer learning was utilized to integrate these models.

After making multiple object detection deep learning models to detect road objects, these models were ensembled to increase the Mean Average Precision (mAP). On top of this ensembled model, a second stage of stacking was performed to add more classes to the existing list of objects. Thus, the model is able to detect all the pavement markings, traffic signs, pedestrians, traffic lights, trucks, dogs, cats, trains, buses, and all the mobile objects that are found on the road.

III. **Outcomes**

Deliverables for this project includes a comprehensive model and a demonstration with test footage in real time. The comprehensive model is based on several deep learning models for autonomous driving, with new functions of pavement markings detection and classification added using transfer learning.

Pavement markings detection and classification is the main focus of this project. The comprehensive model is able to take video footage as input, analyze each frame, detect pavement markings, classify the pavement markings, and then report the findings to assist drivers or autonomous driving systems in making decisions. Detecting and classifying pavement markings allow drivers and autonomous driving systems to drive on the road safely.

The comprehensive model is built upon several existing models, thus it has the inherent ability to detect and classify vehicles on the road, traffic lights, and pedestrians. At the time of writing, the missing link for existing research is the detection and pavement markings. With this project, the missing link is fulfilled, and level 5 autonomous driving will be one step closer to fruition. The comprehensive model is also able to detect different environments such as urban and rural areas, school zones, airports and so on. The model is also able to function well in various weather conditions, such as rain, fog, snow and so on.

At the completion of the project, a live demonstration will be presented. The comprehensive model will analyze a test video footage and showcase results in real time. Model performance is then measured in terms of pavement markings detection accuracy, classification accuracy, and any delays in detection.

IV. **Applications**

The research on object detection, classification and tracking has seen a lot recently. However, their focus does not solve the problem of recognizing a signal on traffic lights or a stop sign. The goal of our project is to deliver a recognition system which not only has object detection but also has the ability to approach the traffic signs or road-marks which advances the object-based recognition system to be more robust to handle real-world transportation. The advanced system we build can help to improve the safety issue of self-driving vehicles which enable the vehicle to react and properly respond to the situation on different traffic signs or road marks. The main usage of our system is aimed at the driving safety risks that we hope to reduce the possibility of potential car crashes due to not having the complete function of the autonomous driving system on recognizing the traffic signals or detecting a specific road mark. We also hope the system can help to reduce the net impact on etrex traveling due to inaccurate decisions given to the vehicle.

ACKNOWLEDGMENTS

We wish to express our deepest gratitude to our committee for their advice and guidance to lead us to work through this challenging and meaningful project. We would like to particularly thank our dear supervisor, Dr. Zeyu Gao for suggesting this wonderful topic and refining our thoughts throughout the project duration to improvise the project plan. With his wonderful instruction, we were able to improve our work. Thank you for putting a lot of effort in hosting presentations and giving feedback to us. We deeply thank you for giving us your precious time and effort in reviewing, editing and listening to our ideas and progress with long patience and endurance. We are very grateful to receive your constant support.

Lastly, we would like to extend our thanks to program Director Dr. Lee Chang for supporting us with important hardware accessibility throughout this journey. Without your help, we would not have been able to complete this project with no costly concerns.

TABLE OF CONTENTS

1. Introduction

1.1 Project Background and Execute Summary

With the rapid development of the economy in modern society, automobiles have become a necessary means of transportation in our daily life. Along with the convenience brought by automobiles, comes the growing traffic safety concerns. In 2020, traffic accidents have been the eighth leading cause of death for children and teenagers who are 5-29 years old. Everyday, around 3,700 people lose their lives in road accidents worldwide. On average, someone dies every 24 seconds on the road. According to the National Highway Traffic Safety Administration (NHTSA), distracted driving is the leading cause of car accidents and truck accidents in the U.S. Distracted driving is difficult to prevent because drivers' attention is easily distracted. Distractions such as diverting attention to children or pets in the car, talking to passengers, or a mobile phone call can cause a driver not to pay attention to road lanes, traffic signs or pedestrians. To aid drivers remain alert at all times, it is crucial to have a driver assistance system which can automatically detect and track traffic signs, lane marks and pedestrians.

With the fast increase in computer processing speed and recent boom in deep learning technologies, many researchers tried different methods to solve traffic sign recognition problems. The problem can be divided into two phases: traffic sign detection and recognition technology. Traffic sign detection is based on extracting the intrinsic characteristics of traffic signs, such as shape, color and texture features. In 2014, Wang et al. designed a red bitmap model to recognize traffic signs. Firstly, they performed color segmentation of input images, and secondly, they performed shape detection of interest based on edge information on the detected images. This model achieved a fair result, but it has a critical limitation. This model can only detect and classify red circular traffic signs. In 2015, Hechri et al. proposed a template matching model to match the traffic signs. They set the sliding window into the same size as the traffic signs, and therefore cut off the useless parts of non-traffic signs in the input image (road scenes). This model achieved good accuracy, but because the real life driving environment is always complex and changeable, this model cannot detect traffic signs in real-time. In 2017, Xiao et al. began a new attempt. They combined HOG features and Boolean convolutional neural network to implement a traffic sign detection method. This method is good at removing the error detection areas of input images, and reaches good performance on traffic sign detection.

On the other hand, traffic sign recognition technology is focused on analyzing, extracting, and classifying detected traffic signs. In 2012, Sun et al. built a traffic sign classification model on the basis of an extreme learning machine (ELM). ELM is a supervised machine learning algorithm related to feedforward neural networks. The algorithm recognized traffic signs based on feature selection and distance calculation results. The model has only one hidden layer, and therefore there were only a few parameters, and the training time was relatively short. This model achieved a fair recognition accuracy compared to other models in that time. In 2015, Qian

et al. used regional depth convolutional neural networks (CNN) to train the model and achieved better recognition rate. In 2016, He et al. designed a ResNet network on the basis of residuals concept. By incessantly learning the residuals, the network gradually achieved better recognition accuracy. In 2017, Yuan et al. combined the support vector machine (SVM) algorithm with Adaboost. The input images were firstly screened by Adaboost, and then recognized and classified by the SVM. The recognition accuracy was very high, but this model takes a long time and calculation power to train.

Apart from traffic signs, road marking is also a crucial visual signal for drivers, especially in structured environments like urban roads and highways. Road markings play an important role in the driver assist system, but very limited research and models are published. Road markings detection is challenging because of the changing road scenes, degradation of the road markings, different lighting situations and weather. In 2016, Xiao et al. designed a structured random forest model to extract a label patch out of each input image. This model is capable of extracting the contextual information of the images, and is able to exploit the structural information of the label to reduce ambiguity. It achieved over 70% recognition accuracy of road markings.

In this project, an improved deep learning model will be proposed. Our model not only detects, recognizes, classifies and reports traffic signs, vehicles, pavement markings and road markings in real time, but also provides drivers with integrated self-driving assistance and solutions. Besides, our model is also capable of detecting the driving environment such as school district, residential area, parking lot and airport, etc.

1.2 Project Requirements

The aim of our project is to apply current state of the art deep learning algorithms on the newly created road markings dataset and develop a model that can detect and track road markings. The model is then combined with other pre-trained deep learning models to build a comprehensive model that can detect, recognize, and classify road markings, traffic signs, and vehicles in real time. The model can be integrated into autonomous driving systems to provide information and assist in decision making. In order to achieve our goal, the following functional and AI-powered features requirements should be met:

1. Functional Requirement:

- Collect and create a dataset of image or video files as input.
- Train an efficient model for recognizing and classifying road markings, traffic signs, and vehicles in different weather conditions.
- Build a real-time output system that can send users or autonomous driving systems information based on the recognition results.

- Have a mock test environment with mimicked real-world driving environment to test the performance of our model
- Accomplish functional requirements 1-4 with only a vehicle camera and related data augmentation technologies and computer vision software.

2. AI-powered Feature Requirements:

- Accurately extract color, size, shape, and edge information of the input images extracted from the videos captured by cameras mounted on vehicles.
- Obtain at least 75% accuracy from the traffic sign, road markings and weather condition detection model, and give users accurate and timely driving guidance. A threshold will have to be optimized for when to accept the detection of an object.
- Increase computational efficiency by selecting the minimum necessary input features (parameters) and constructing proper layers of the deep learning model.
- Reduce latency between computation and action of the driver guidance system and achieve real-time driver assistance system.

1.3 Project Deliverables

Deliverables for this project include a comprehensive model and a demonstration of comprehensive model processing test footage and generating results in real time. The comprehensive model is based on several deep learning models for autonomous driving, with new functions of pavement markings detection and classification added using transfer learning. Road markings detection and classification is the main focus of this project. The comprehensive model is able to take video footage as input, analyze each frame, detect pavement markings, classify the road markings, and then report the findings to assist drivers or autonomous driving systems in making decisions. Detecting and classifying pavement markings allow drivers and autonomous driving systems to drive on the road safely. The comprehensive model is built upon several existing models; thus, it has the inherent ability to detect and classify vehicles on the road, traffic lights, and pedestrians. At the time of writing, the missing link for existing research is the detection and pavement markings. The comprehensive model is also able to detect different environments such as urban and rural areas, school zones, airports and so on. The model is also able to function well in various weather conditions, such as rain, fog, snow and so on.

The goal of our project is to deliver a recognition system which not only has object detection but also has the ability to approach the traffic signs or road-marks which advances the object-based recognition system to be more robust to handle real-world transportation. The advanced system we build can help to improve the safety issue of self-driving vehicles which enable the vehicle to react and properly respond to the situation on different traffic signs or road marks. The main usage of our system is aimed at the driving safety risks that we hope to reduce

the possibility of potential car crashes due to not having the complete function of the autonomous driving system on recognizing the traffic signals or detecting a specific road mark. We also hope the system can help to reduce the net impact on etrex traveling due to inaccurate decisions given to the vehicle.

At the completion of the project, a live demonstration will be presented. The comprehensive model will analyze a test video footage and showcase the detection results. Model performance is then measured in terms of pavement markings detection Mean Average Precision (mAP), classification accuracy, and any delays in detection.

1.4 Technology and Solution Survey

In order to detect, recognize and classify the traffic signs, road markings and other traffic control devices accurately and timely, deep learning algorithms combined with GPU are considered to be the best way. Below are the comparison table of different deep learning model structure, advantage, disadvantage, and important features of the most commonly used deep learning algorithms in this area:

Deep Learning Models	CNN (GTSRB)	Support Vector Machine	Classical LeNet-5 Network	YOLO v2
Objective	Recognize and classify 11 kinds of traffic signs images	Detect and classify 9 categories of road markings under various complex road conditions.	Traffic sign and road markings recognition	Real-time detection and classification of traffic signs
Model Structure	Three convolutional layers, one activation layer (ReLU), one fully-connected layer and one output layer. Learn in a backward optimization way.	Two SVMs. SVM1 outputs two categories of information: Road markings and Road Background, Road markings will be input of SVM 2, and SVM2 output category 1-9.	Two convolutional layers, two pooling layers, two fully-connected layers and one output layer	Bottom Layer: Conv3-32. Network structure: 4 Maxpool/2 layers, 22 convolutional layers, 2 route layers and 1 detection layer
Recognition Accuracy	99.7%	97.4%	84.8%	90.37%

Advantage	Model learn features from a large number of samples without preprocessing, fast training time, and avoids the design difficulty of hand-crafted features	Good classification performance on all kinds of road markings and road background. Robust model.	Good classification and recognition effects for a single target	Achieve real-time detection and fast classification
Disadvantage	Because of complex computation, performance of real-time prediction is not good.	Not able to perform in real-time.Requires lots of image preprocessing and postprocessing work, such as inverse perspective transformation, dimensionality reduction, candidate region extraction, feature extraction, etc.	Training network cannot converge, and the recognition efficiency of the network decreases dramatically	Classification and Recognition accuracy can still be improved
Real Time Prediction	No	No	No	Yes
Distance Calculation	Yes	Yes	No	Yes

Table 1.4.1 Comparison of most commonly used deep learning algorithms in traffic sign and road markings detection area

1.5 Literature Survey of Existing Research

Models for detecting road traffic objects have been developed for many years since the concept of self-driving cars came into the picture. These models were generally made for detection of traffic lights, signs, lanes, or vehicles. Replacing previous object detection models with its accuracy, convolutional neural networks (CNN) is dominant in this research domain as well. There are various models used for traffic object detection and tracking including Faster R-CNN, Single Shot Detector (SSD), and You Only Look Once (YOLO). These methods are highly used by researchers for object detection and tracking.

A similar real time traffic sign recognition model is applied in Rajendran et al (2020) by implementing YOLOv3 (YOLO Version 3) which is a recently developed state of the art method. The table below provides comparison and ideal conditions for working of these models. An application of multitask Faster R-CNN by Yang et al (2020) is seen in [14] where the researchers

were successfully able to train a model to detect vehicles in the images. Similarly, Jamiya et al (2021) has applied Little YOLO (YOLOv3 tiny) for real time vehicle detection. For lane detection, Muthalagu et al (2020) has applied Color Segmentation and Perspective Transformation for Advanced Driver Assistance System (ADAS). After detection of the traffic signs using Single Shot Detector (SSD), Campbell et al (2019) also calculated the distance of these signs from the camera.

Table 1.4.2 below shows the comparison of past research in traffic object detection and tracking using various metrics.

Metric/ Research Paper	**Real Time Traffic Sign Recognition using YOLOv3 based Detector.**	**LittleYOLO-SPP: A delicate real-time vehicle detection algorithm.**	**Lane detection technique based on perspective transformation and histogram analysis for self-driving cars.**	**A multi-task Faster R-CNN method for 3D vehicle detection based on a single image.**	**Detecting and mapping traffic signs from Google Street View images using deep learning and GIS**
Objective	Real time traffic sign recognition	Real time vehicles detection	Real time Traffic Lane Detection	Image based Vehicle Detection	Image based traffic sign recognition
Model	YOLOv3	YOLOv3-tiny	Color Segmentation	Faster R-CNN	Single Shot Detector
Training and Evaluation Data	Training and evaluation are done using German Traffic Sign Detection Benchmark and classifier performance is verified using German Traffic Sign Recognition Benchmark.	PASCAL VOC 2007, PASCAL VOC2012, MS COCO 2014 for training and GRAM Road-Traffic Monitoring (GRAM-RTM) video dataset for detection result analysis.	Video data collected by using a camera fixed to the front of the vehicle.	The KITTI Dataset	Google Street View Images

Accuracy	92.2%	77.44%	RMSE value of 5.5126 and accuracy of 95.75%.	83.5%	95.63%
Real time Prediction	Yes	Yes	Yes	No	No
Distance Calculation	No	No	No	No	Yes

Table 1.4.2. Comparison of past research work for traffic object recognition

Table 1.4.2 compares the various algorithms that can be used for real time traffic object detection. All these models are the state-of-the-art techniques used currently by researchers as shown in table 1.4.1. These techniques have various advantages and disadvantages over others. A comparison using various metrics can be found below.

Model/Metric	Accuracy	Speed	Small Object Detection
Fast R-CNN	High	Low	Yes
Faster R-CNN	Medium	Medium	Yes
SSD	Medium	Medium	No
YOLOv1	Low	High	No
YOLOv3	High	High	Yes

Table 1.4.3. Comparison of Object Detection Algorithms

All the models shown in table 1.4.2 are used for real time traffic object detection considering the requirements of the models. Faster R-CNN gives the highest accuracy, but it is not good at speed. Hence, researchers prefer the recently developed YOLOv3 for real time detection.

From all this research work, it is clear that tremendous research has already been done in this domain. However, none of the models were trained to detect road markings which will be included in this research to fulfill the gap. An integration of all this research work to develop a single model for detecting all traffic objects will be delivered as the outcome of this project. The project also considers calculating the distance of the detected objects. Zhang et al (2018) has implemented a method called binocular stereo vision based on deep learning for the distance calculation. The method has significant results and can be applied in other distance calculation applications. Campbell et al (2019) has implemented the photogrammetry approach for distance calculation after detection of traffic signs in images. This method uses camera focal length, pixels of the traffic objects in the image and actual dimension of the object. With predefined dimensions of traffic signs, and road marks, distance can be calculated for real time application involving simple calculations. With all these deliverables, the scope of this research is extended far beyond the existing research work.

2. Data and Project Management Plan

2.1 Data Management Plan

Data for this project mainly consists of driving video footage, and static images converted from videos. The main challenge regarding data is availability. We could not find a dataset that is dedicated to pavement markings. Therefore, we collected data to satisfy our topic and process existing datasets to extract additional information. Our data management has four parts: data collection, data management, and storage.

Data collection is the second most labor-intensive task in data management. Existing datasets online are not dedicated to pavement markings despite the abundant amount of road signs data available online. To build a dataset that has many images for all types of pavement markings, we use a dash cam to record video footage driving in town and freeway. Since we cannot possibly drive through all the roads in the United States to cover all pavement markings and driving conditions, we will use Google Maps to capture images of pavement markings.

Data management is the most labor-intensive task, which consists of reviewing all new and existing data, organizing and labeling, and annotation. For collected data, all videos are reviewed, and the frames that contain pavement markings are extracted and converted to images. To ensure our data covers all types of pavement markings, and diversify the road conditions, we used Google Maps to capture images at different locations. All images have at least 800 by 600 pixels to maintain a reasonably high resolution while reducing computational requirements later in model development. Images are annotated to have labels for different pavement markings, cross traffic, road signs and lights, and vehicles. This is done with various tools such as MakeSense.ai, and we use bounding boxes, lines, and polygonal segmentation depending on the

shape of an object of interest. Moreover, our goal is to create a comprehensive model that can generalize well and detect pavement markings in different weather and lighting conditions, and augmenting a dataset with artificially created weather and lighting conditions will help our model achieve better model generalization.

Regarding storage, we mainly stored data on Google Drive. Team members have access to all data any time. Because there is no sensitive information involved, data security and access are not of concern. Google Drive also enables us to utilize Google Colab and seamlessly read data stored on Google Drive. Team members can also download data to local computers if they prefer to train models with a dedicated GPU.

2.2 Project Development Methodology

Project development is split into two phases: primary scope and expansion. We strive to achieve the primary scope, and then expand the scope and add more functions to expand the comprehensive model. Development takes time, labor, and computational resources; and these are major constraints we must account for. Thus we are implementing constraints reviews throughout the development process. As the set date of final demonstration nears, we have a limited time frame for each step of the process. If one step takes longer than expected, we need to make adjustments for the following steps accordingly. Same applies to labor and computational resources. If constraints are tight, we may need to work with a smaller dataset with fewer annotations or smaller models that require less powerful computing hardware and less time for training.

There are two constraint review steps in our project development cycle primary scope phase, as shown in figure 1; one is after the initial data process, and the other after initial model performance evaluation. In the expansion phase, we follow the development cycle in figure 1 and have two constraint reviews. The expansion phase follows the development cycle except for topic selection.

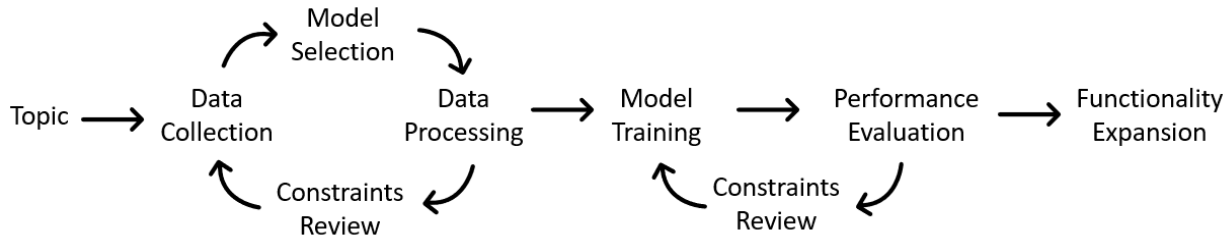


Fig 1. Project Development Cycle

2.3 Project Organization Plan

During project planning, the cross-industry standard process for data mining (CRISP-DM) was referenced, and the project is broken down to the following phases: problem understanding, data collection, data processing, modeling, evaluation, and demonstration. While the project planning is derived from CRISP-DM, it differentiates for it is moving forward instead of being a cycle like the CRISP-DM. After the topic, goal, and steps are finalized, the project moves forward and proceeds to follow the development cycle detailed in section 2.2 above.

At the beginning of project planning, several potential topics were discussed, and the topic was narrowed down to object tracking related to autonomous cars. After research on recent publications regarding vehicles, pedestrians, and road signs tracking, authors discover that very few, if any, research papers studied pavement markings and decide to adopt it as the topic. Since the topic is unique, there are no existing datasets that perfectly match the criteria and authors have to collect data and create a new dataset that contains pavement marking images and annotate the images accordingly. For modeling, models from previous research are referenced and trained using the new datasets. After the initial models are evaluated, transfer learning is applied so that the comprehensive model will expand in functionality and be able to classify and track other objects such as vehicles, pedestrians, and road signs.

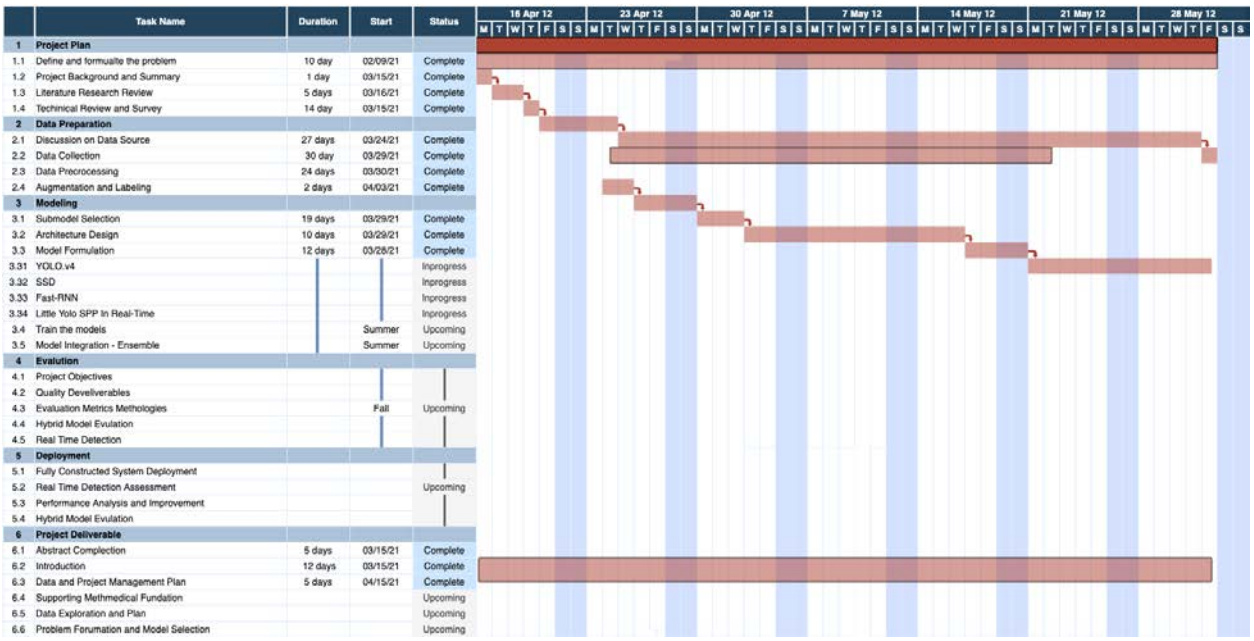
2.4 Project Resource Requirements and Plan

Deep learning is very computation-intensive, which undoubtedly requires multi-core and high-speed CPUs. In order to help us to train our Faster R-CNN and YOLOv4 we performed on the cloud platform called Ucloud. The GPU-enhanced cloud host is based on the mature cloud computing technology of UCloud, which greatly improves the capabilities of graphics and image processing and high-performance computing, and has the characteristics of flexibility, low cost and ease of use.Effectively improve the processing efficiency of graphics processing, scientific computing and other fields, and reduce IT cost input.

Currently, Ucloud provides G1 (Tesla K80), G2 (Tesla P40) and G3 (Tesla V100) hardware models, as well as T4S, 3090, V100S and other models which is significantly helpful for large-scale offline computing and AI training. And the EPC which is a high performance computing cluster that has a master frequency up to 3.5GHz, CPU up to 128 cores, memory up to 512Gb which is applicable to our needs. Locally we use Tensorflow 2.0 or Pytorch 1.5 for the model construction to prepare training and testing. The whole code is written in python 3.8. However, we do not know the exact potential cost. In addition to that, we have also utilized Google Colab Pro for computations throughout the project timeline. Colab Pro costs $ 9.99 each month and each team member used Colab Pro for 4 months which cost us approximately $160.

2.5 Project Schedule

In order to demonstrate the project in a clear and concise manner we decided to use the Ganntt chart to provide a fully written schedule to follow up. At the beginning of our Real Time Object detection project, we need to collect the road mark and line mark image hand by hand from Google and then augmented to make at least 1000 images for each individual mark. Our research includes model selection and architecture design have accompanied the whole project to guarantee the best results and success of the project through the incorporation of new ideas and solutions to the different phases. The way we manage our team work is to divide the “dirty” image augmentation work by splitting it into subgroups and each one of our team members would work on each subsection independently and combine all the work as a hybrid model. So after breaking down the work, we followed each task on the chart on time. The blue color indicates the completed section till March 2021 and green represents the upcoming tasks that we worked on till December 2021.



	Task Name	Duration	Start	Status
1	**Project Plan**			
1.1	Define and formualte the problem	10 day	02/09/21	Complete
1.2	Project Background and Summary	1 day	03/15/21	Complete
1.3	Literature Research Review	5 days	03/16/21	Complete
1.4	Technical Review and Survey	14 day	03/15/21	Complete
2	**Data Preparation**			
2.1	Discussion on Data Source	27 days	03/24/21	Complete
2.2	Data Collection	30 day	03/29/21	Complete
2.3	Data Precrocessing	24 days	03/30/21	Complete
2.4	Augmentation and Labeling	2 days	04/03/21	Complete
3	**Modeling**			
3.1	Submodel Selection	19 days	03/29/21	Complete
3.2	Architecture Design	10 days	03/29/21	Complete
3.3	Model Formulation	12 days	03/28/21	Complete
3.31	YOLO.v4			Inprogress
3.32	SSD			Inprogress
3.33	Fast-RNN			Inprogress
3.34	Little Yolo SPP In Real-Time			Inprogress
3.4	Train the models		Summer	Upcoming
3.5	Model Integration - Ensemble		Summer	Upcoming
4	**Evalution**			
4.1	Project Objectives			
4.2	Quality Develiverables			
4.3	Evaluation Metrics Methologies		Fall	Upcoming
4.4	Hybrid Model Evulation			
4.5	Real Time Detection			
5	**Deployment**			
5.1	Fully Constructed System Deployment			
5.2	Real Time Detection Assessment			Upcoming
5.3	Performance Analysis and Improvement			
5.4	Hybrid Model Evulation			
6	**Project Deliverable**			
6.1	Abstract Complection	5 days	03/15/21	Complete
6.2	Introduction	12 days	03/15/21	Complete
6.3	Data and Project Management Plan	5 days	04/15/21	Complete
6.4	Supporting Methmedical Fundation			Upcoming
6.5	Data Exploration and Plan			Upcoming
6.6	Problem Forumation and Model Selection			Upcoming

Fig 2.5.1 Gantt chart for Project Scheduling

3. Data Engineering

3.1 Data Process

The aim of our traffic recognition and guidance system is to accurately detect and classify various types of road markings, cars, pedestrians, traffic signs along with other mobile objects on the road and provide drivers with real-time guidance. In order to achieve that, we developed the following data process workflow to collect, preprocess and transform our dataset:

1) Collect traffic sign data and divide them into training, testing and validation dataset
2) Collect road markings dataset
3) Perform data statistics on step 1 and step 2 for further processing
4) Collect more raw image data for road markings by vehicle camera, google images, google earth, etc. to solve the dataset imbalance problem
5) Collect vehicle, pedestrian, traffic lights, traffic signs, and other mobile objects data to use transfer learning with the COCO model.
6) Data preprocessing: resolution unification and image sample selection
7) Data transformation: Image data augmentation, etc.
8) Data integration: Combine the clean image data and form our own training, testing and validation dataset

We spent a lot of time on step 1 to 6 due to the vital role data collection and preprocessing plays on the final performance of deep learning models. An image dataset with good quality can help better capture the key features, reduce the training time, and improve the accuracy and robustness of deep learning models.

For step 1, because there are some available public datasets that contain traffic sign images, we primarily compared the quality and resolution of different datasets, and carefully selected the ones that mostly fit our task and required fewer preprocessing steps. However, for road markings images, public resources are relatively less. Therefore, in addition to research open and available datasets, we developed different kinds of methods to collect our raw dataset. We will discuss this part in detail in 3.2.3.

3.2 Data Collection

3.2.1 Traffic Sign, Vehicles and other Mobile Objects Dataset Collection

For the pretrained classes to merge with roadmark dataset, after researching and comparing different open datasets including Kaggle dataset, the computer vision laboratory dataset, laboratory for intelligent & safe automobile (LISA) traffic sign dataset, and MTSD dataset, Udacity self driving dataset etc. we decided to primarily make our dataset for these additional classes. We used the Udacity self-driving dataset to collect images for training and edited all the annotations to meet the class requirements for our model.

The Udacity self-driving dataset has 11 classes and has 30,000 images collected during different weather and times of day. Image data were obtained from different cameras, and image sizes are from 640*480 to 1024*522 pixels. Traffic sign and other objects sizes varies from 6*6 to 167*168 pixels. In data selection process, we filtered those images that are in grayscale, and only kept those in RGB, and those were able to be converted into 800*600 pixels. We used python and other tools to make sure all the data are annotated with sign type, position, size, whether occluded, and whether on the side road.

Besides the collection of images, the label list was changed from 10 classes to 5 classes for all the image labels. The previous annotations had classes such as biker, car, pedestrian, truck, and 7 different classes for traffic light based on colors and direction of signal. All the traffic lights were changed to one single class and the total classes from Udacity dataset were changed to 5. There were 23 roadmark classes decided to collect for our model. Hence, all the YOLO annotation files were changed to follow label-number 23-27 for the udacity classes to follow along with the roadmarks. In parallel with this collection process, we were also collecting the roadmark classes.

Once, we collected the images and labels for pre-trained classes as well as roadmarks, we added 5 more classes for our model which made the total classes to 32. However, we did not have annotations for these new classes. We also lacked labels for roadmarks in the Udacity classes and lacked labels for the pretrained classes in the roadmark dataset that we collected. This was leading to problems in retaining the weights during training. Hence, labelled all the images in the roadmark for the additional Udacity classes so that the model does not forget the weights on training. Similarly, we also labelled all the images in the Udacity dataset for roadmarks making a one complete dataset. This task was very tedious and again took a lot of time.

3.2.2 Road Markings Dataset Collection

It is relatively hard to find a comprehensive dataset of road markings that has already been summarized and annotated. Therefore, for the road markings dataset, we decided to build our own dataset in this way: 1) Research and select a small dataset as our basic dataset 2) Analyze the data statistics of our basic dataset. Filled, searched and collected some certain types of road markings (such as ‘school district’, ‘Keep’, ‘40’, ‘45’, ‘50’) that are very sparse in our basic dataset.

For our basic road markings dataset, we chose the dataset collected by Tao W. and Ananth R. (2012), which contains 1,400 labeled images that are captured from a set of videos taken by a vehicle camera. Since the images are all taken by same device with the same 800*600 resolution, therefore it contains rare noises. However, after analyzing this basic dataset, we found that the data is highly imbalanced. Among 1400 images, 705 images are in ‘Left Turn’ type, 112 images are in ‘35’ type, 101 images are in ‘Right Turn’ type, but there are only 7 images in

‘Yield’ type, and only 2 images in ‘30’ type. We decided to fill those sparse categories with raw images we collected from different methods.

3.2.3 Raw Image Dataset Collection for Road Markings

It took us lots of effort and time to collect raw images. We primarily collected our raw image dataset using the following methods: 1) Use Google Earth birds eye view on streets in the U.S. After finding the types of road markings we are looking for, we switch to the driver mode viewpoint and take a screenshot of the image. 2) Use street view in google map to search our neighborhood for target road markings; after finding it, we either take a screenshot of the street view or drive to that place and take pictures of the road marks using our dash cameras. 3) Use web crawlers and scraping techniques to search and collect images from the internet, and filter them manually by size, image quality and viewpoint. We principally collected raw dataset in the following road markings categories: ‘Diamond’(carpool sign), ‘School District’, ‘Xing’, ‘Ped’, ‘Bus’, ‘Forward’, ‘Stop’, ‘Bike’, ‘45’, ‘50’, ‘Clear’, ‘Forward & Right’, ‘Forward & Left’. The current target of our raw dataset is to collect 300 images for each category of road mark, so our dataset will be balanced, data preprocessing and transformation workload will be minimized in the future, and the final performance of our deep learning models will be improved.

Fig 2.3.1 Sample image data from multiple sources

3.3 Data Preprocessing

In the data pre-processing step, we carefully filter those images which are far below our target resolution size. As models like YOLO that we are implementing require input image size in multiples of 32 pixels, we have preprocessed all of our data and converted them to dimensions of 480x480 pixels. All used images are in a three channel RGB color space, so grey-scale images

were removed from the dataset. Besides, images with low fidelity or other errors were removed as long as we encountered and discovered them. In order to unify the image resolution for the whole dataset, we used Keras Preprocessing library in Jupyter Notebook. Besides, we used an annotation tool named 'Labelme' to manually add rectangle shapes and label names to the road mark/traffic sign on each image. For example, Figure 2 shows each image containing a bounding box around the road marking area. Different colors of the bounding box indicate different types of labels on images.

In addition to this, more data was generated from images with many road marks by zooming in, translating in X-Y directions, rotating the images, as well as flipping them. All these basic data augmentation techniques increased the size of our dataset. More advanced augmentation techniques were also performed which are discussed in the next section. At this stage, the resolution of the images were changed to ready to train model requirements.

Fig 3.3.1 Sample labeled data

3.4 Data Transformation

Data transformation is necessary for this application as the road markings detected live will be in different conditions. To achieve these various types of images, we are implementing data augmentation techniques. A good model should be able to detect the road markings in all different scenarios. Hence, we are generating new images from our existing labeled and unlabeled data. These scenarios can be due to weather, time of the day, lighting conditions, speed of the vehicle, or objects on the road. On implementing the model, these situations will always

come up when recognizing the road markings in a live video. For data augmentation of the road images, we are using the Automold Road Augmentation library in Python to get all the effects mentioned above.

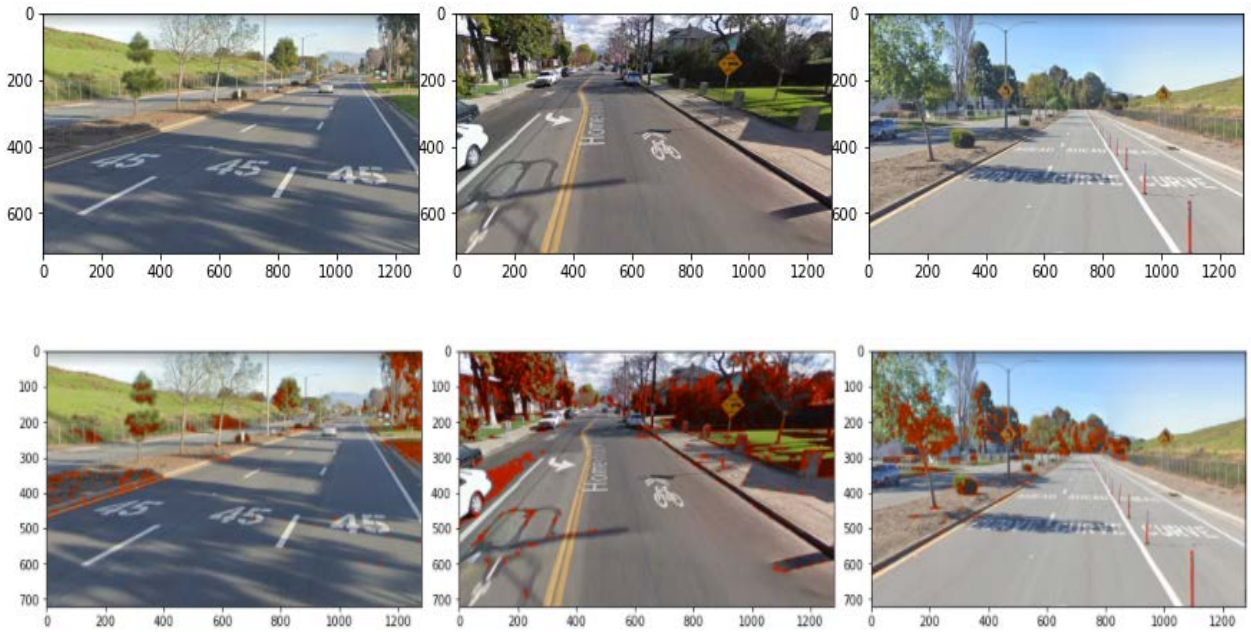


Fig 3.4.1 Autumn effects using Data Augmentation

3.4.1 Weather conditions

To overcome the weather problems for our model, we are augmenting our images by adding various weather effects. We are creating new images by adding the presence of snow, rain, fog, and leaves in the images. These weather conditions lead to partial visibility of the traffic objects and hence can lead to deviation in detecting the required objects. For example, snow on the road with white pixels can be deviating when it is near pavement markings. Amount of snow can be controlled by the coefficient ranging between 0 and 1 where 0 is the original image and 1 making the full picture with snow pixels. Similarly for rain, the slant of the rain drops, drop width, drop length, drop color, and the type of rain can be given as inputs randomly during night and day images to generate all the different conditions. Similarly, fog coefficients and autumn coefficients have been added to generate such conditions on the road.

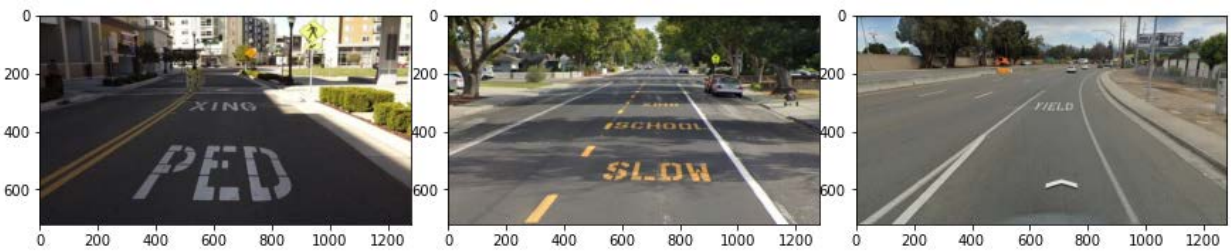


Fig 3.4.2. Rain effects using Data Augmentation

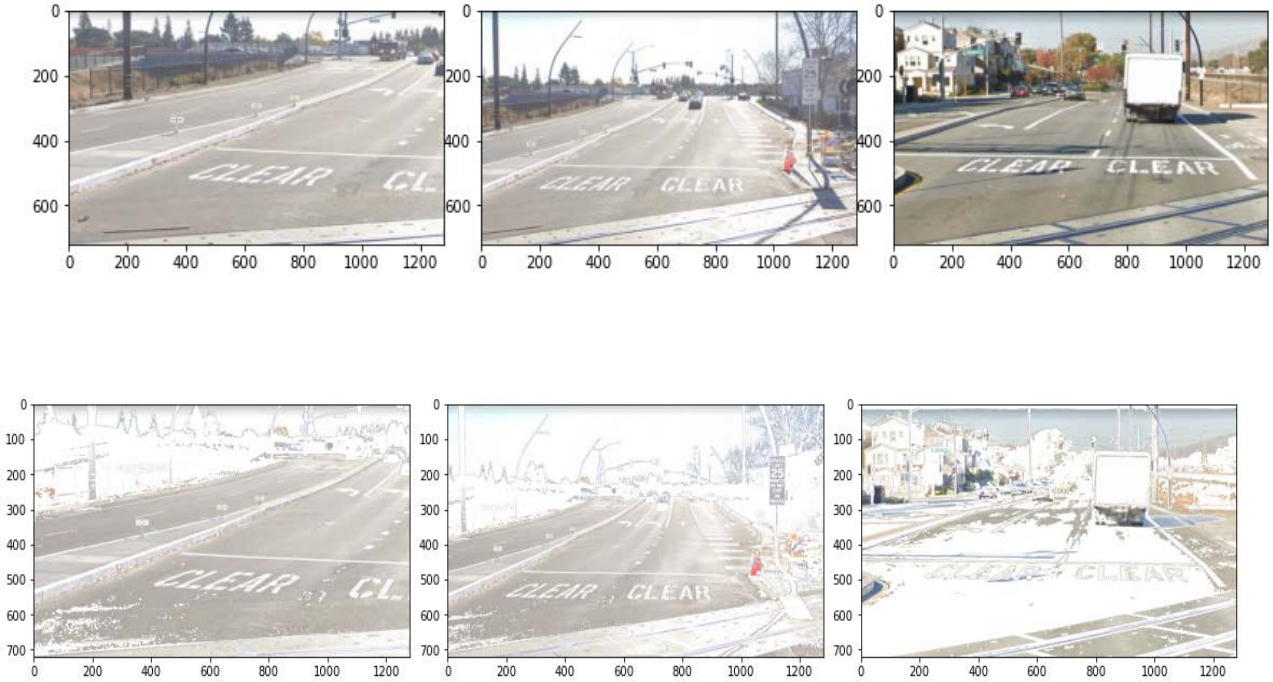


Fig 3.4.3 Snow effects using Data Augmentation

3.4.2 Brightness, contrast, and darkening effects

We are changing the brightness and contrast of the images to create images for different lighting conditions for different times of the day. Increasing the brightness of images will make the model robust for detecting the roadmarks in the bright sunlight. The amount of brightness added to be added in the image can be varied from intensity of 0 to 1 where 1 being the brightest and 0 is the original image. For some roadmarks and traffic signs, the images are collected only for the day time. These images are converted dark to resemble night time by darkening. Just like

brightness, darkening values range from 0 to 1 where 0 gives output as the original image and 1 gives a full black image.

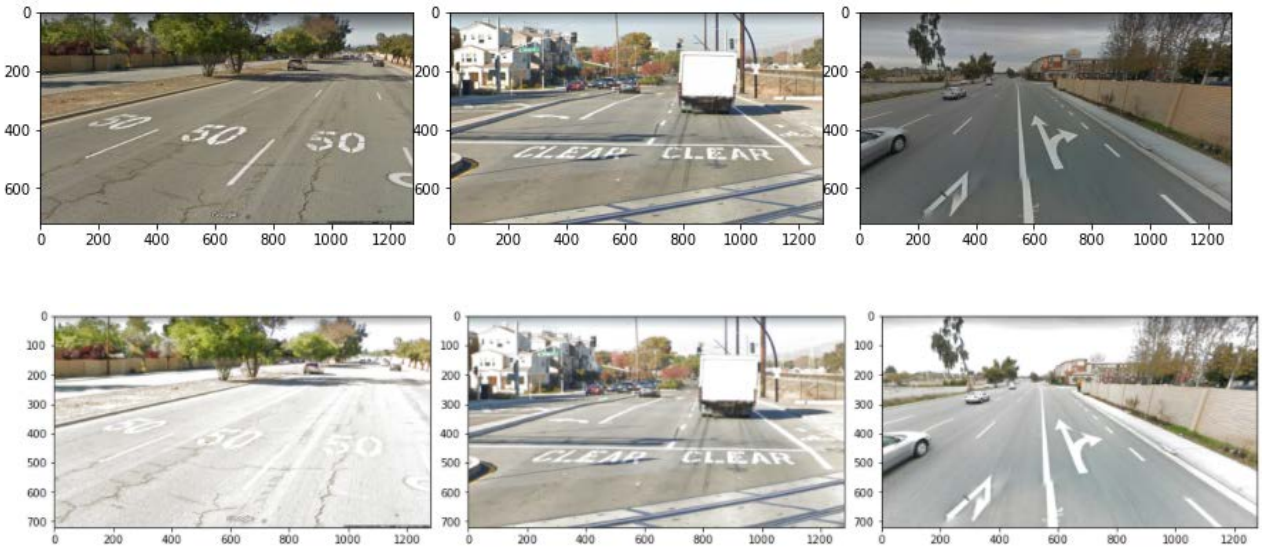


Fig 3.4.4 Brightening the images

3.4.3 Flare and Reflection

In real life scenarios, many times sun flare will come up on the camera glass of the automatic driving system and the model will not be able to capture full roadmark datasets to classify and interpret. Thus, adding images of such effects will make our model more robust. The radius and centre of the flare, the number of flares, and color in RGB were passed to create sun_flare in the images. Sometimes, these flares are not directly generated from the sun but the reflection of sun from shiny roads makes the markings invisible to the camera. Augmented images of such effects are also added to tackle this issue during a live video.

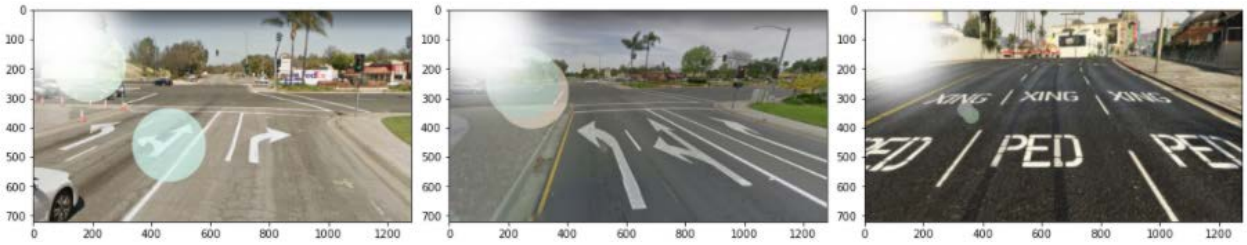


Fig 3.4.5 Adding sun flares on the camera lens

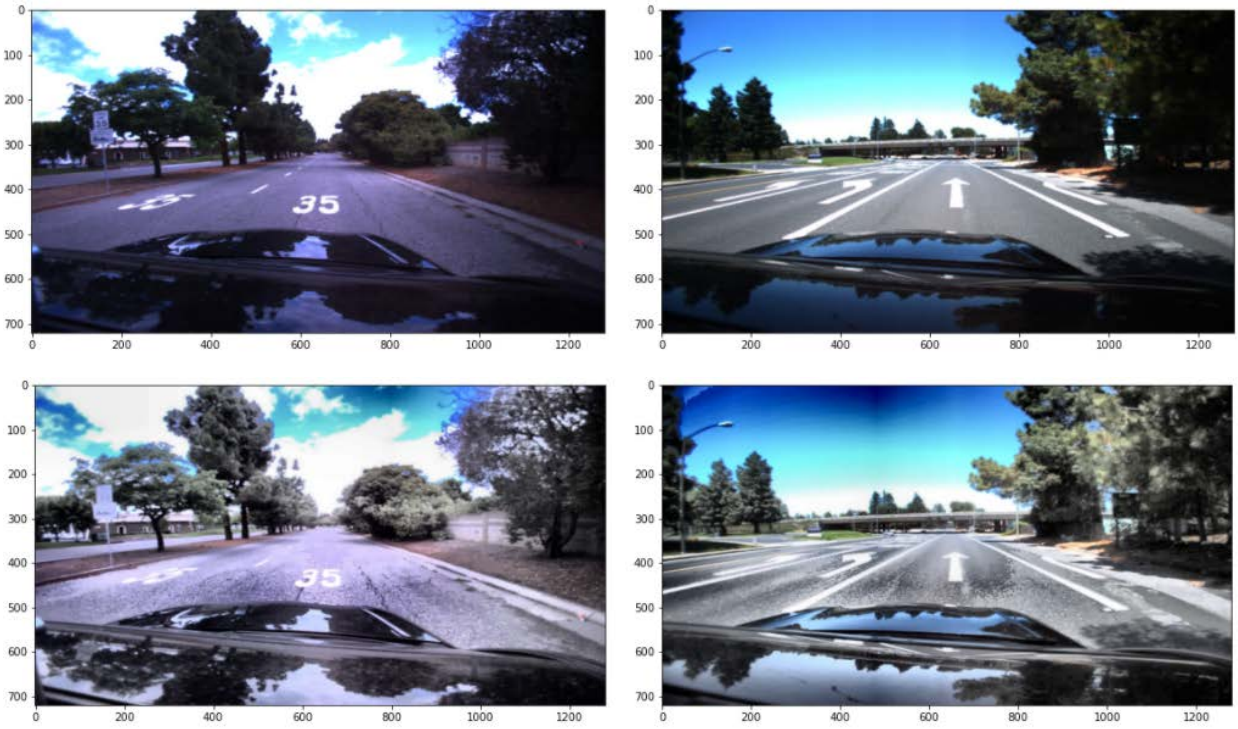


Fig 3.4.6 Correct Exposure of the image

3.4.4 Road objects

Some other data augmentation techniques being performed is to create various road conditions due to objects on the roads and the travelling speed of the car. For example, adding gravel on the roads, adding manholes on the roads, adding shadows in full bright images, and blurring the images to resemble how the captured image might look like when a vehicle is travelling at a high speed.

Fig 3.4.7 Adding gravel patches on the road

For adding gravel, the rectangular region of interest and the number of gravel patches required are given input randomly. For manholes, the elliptical centre, color, height, and width are defined for random images to resemble real manholes on the road. For shadow, the region of interest, and the number of shadow objects are defined randomly for the images. Atlast, few random images from original and augmented images were blurred by defining the speed coefficient for augmentation. All these transformed images will change the pixel intensities of the roadmarks and will avoid overfitting in our model.

3.5 Data Preparation

After adding the augmented images to the dataset, the images will be shuffled and randomly selected for training and testing. For the road marking dataset, we are dividing the labeled data for training, validation, and testing in a ratio of 60:20:20. For the traffic sign images collected from LISA traffic sign data, German Traffic Recognition Dataset Benchmark, and the Mapillary Traffic Sign Dataset along with the augmented images are also divided in the percentage of 60:20:20 for training, validation, and testing respectively.

For the manually collected data, we are currently annotating the images and storing the labels in a .txt file for each image. For YOLO, the annotations are stored in a separate file and multiple labels are written in the next line. The first column represents the class label, second and third column represents x and y points in the image representing the pixel number and the last two columns represent width and height. Each of these values are separated by a space. Figure 12 represents the sample image annotation for YOLO.

```
0 45 55 29 67
1 99 83 28 44
```

Fig 3.5.1 Image Annotations for YOLO

Fig 3.5.2 Training Dataset Example for roadmarks

The above image shows an example of how the training data will comprise all the data in a different scenario. As the model will be implemented to detect traffic road marks in a live video, the model will also be tested on videos collected. The video will help evaluate the working of the road marks detection model as well as the traffic sign recognition model.

3.6 Data Statistics

A prelabeled data for all the road markings in the US is not available yet. To get all the required images, we collected raw data using Google Street View, Google Earth, web crawlers and scraping as discussed in section 3.2. Some labeled images were collected from the Roadmarking dataset. This dataset had 1400 labeled images of different road markings. As per the requirement in the model, at least 100 images were required for each marking after preprocessing and data augmentation to create different scenarios. As the dataset had an imbalanced number of images, 824 images were collected manually using different techniques mentioned above. These manually collected images had different dimensions and interrupting objects in the image that were manually removed. These images are being augmented to generate new images that comprise various road conditions for every road marking. The table below shows the number of images that were pre annotated, and manually annotated. We are still in the process of annotating our data.

Type of Road Marking	Label Name	Label Number	From Annotated dataset	Images Collected and Augmented
25	25	0	15	400
30	30	1	2	400
35	35	2	112	300
40	40	3	69	300
45	45	4	0	400
50	50	5	0	400
Bike	bike	6	41	400
Bus	bus	7	0	300
Diamond (carpool)	diamond	8	1	400
Forward	F	9	80	400
Forward & Left	FL	10	6	350
Forward & Right	FR	11	13	350
Keep Clear	KC	12	12	300
Left Forward Right	FLR	13	0	200
Left turn	L	14	705	450
Ped	ped	15	54	250
Rail	rail	16	90	250
Right turn	R	17	101	300
School	school	18	5	250
Stop	stop	20	49	400
Xing	xing	21	70	250
Yield	yield	22	7	250

Table 3.6.1 Number of road marking images

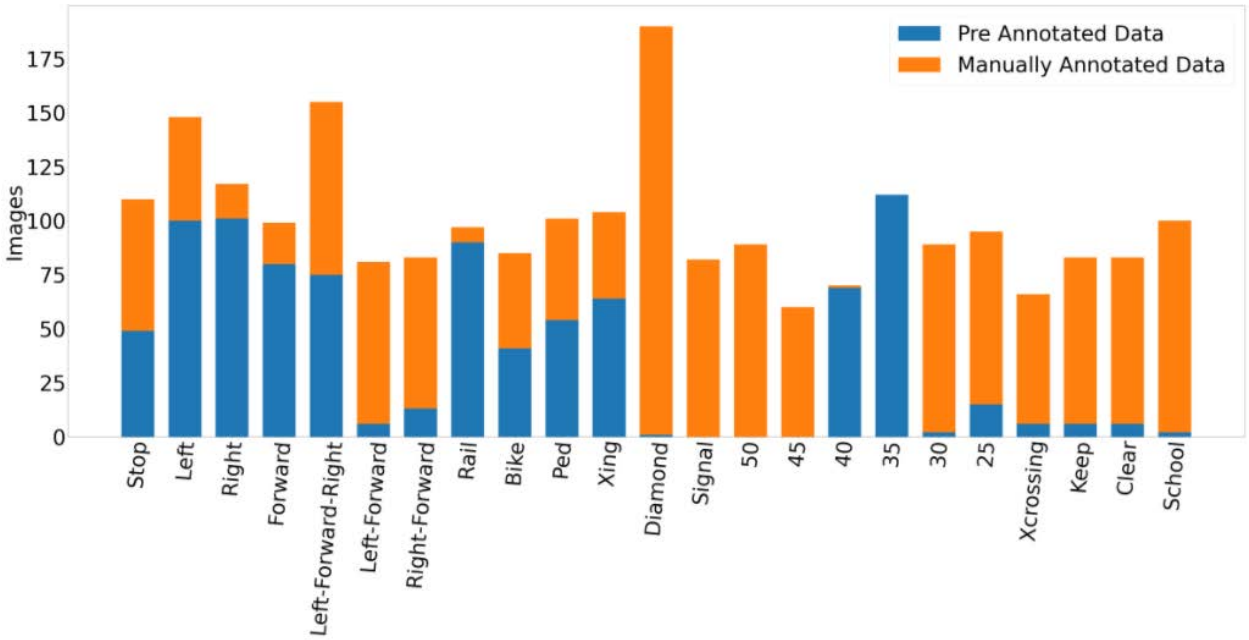


Fig. 3.6.1 No. of roadmark images pre-annotated and manually annotated

In the dataset, left turn images are more than half which were removed as most of the images were captured from one still location and only 100 images were manually selected. For all other markings, few images will be augmented to fulfill the image requirement as well as to remove the imbalance in the data. The stacked bar chart above shows the percentage of total data manually processed and the pre labelled data.

4. Model Development

4.1 Model Proposals

In this paper, five algorithms are examined and trained using our road markings dataset. Algorithms include improved You Only Look Once version 4 (YOLOv4), improved Faster RCNN model, improved single shot detector (SSD), CenterNet, and EfficientDet1 which is also the state of the art model. The accuracy, time to detect, and loss curve in training the model will be compared and the best model will be selected. These four models are selected because each represents the best in class. Faster RCNN is the most accurate thanks to its two stage architecture, SSD is the fastest, YOLOv4 is a good compromise in general, and EfficientDet1 and CenterNet being the latest developed among all these has better efficiency as well as accuracy.

After shortlisting these five models, they will be trained on the roadmarking and the Udacity dataset for 32 classes. Further, these models will be evaluated to compare the mAP, and

detection time. Based on the results, the best four models will be ensembled to achieve better mAP and low processing time. This ensembling will be the Stage 1 ensembling.

a) Two Stage Ensembling model

We propose to perform the ensembling process in two stages. Each stage is responsible for making the model better in different ways. In the first stage ensembling, the output from four different models will be stacked to get high mAP and better accuracy. This will result in an improved model which is robust to disadvantages of different models and overcomes their shortcomings. Similarly, it will also be able to take the benefit of the advantages of different models.

After Stage 1 ensembling, the model will be stacked with another pretrained YOLO model trained on COCO dataset to add more classes to existing roadmark and pretrained classes from Udacity Dataset. After adding 90 more COCO classes to the existing 32 classes of our dataset, the two staged ensemble model will be able to detect almost all the objects on the road including buses, trains, dogs, cats, pedestrians, and even airplanes. The second stage model will increase the applications of our system for various uses.

i) Stage 1 Ensembling

For the stage 1 ensembling, running all the four models together will take a lot of processing power. To overcome this, we intend to generate output files from all the individual models and then ensemble them separately using weighted boxes fusion method. This will ensemble the model with the consensus or maximum vote method resulting in a better output. A threshold for confidence score for each model can be set to check if it should be considered in the ensemble process. Similarly, a threshold value for IoU will decide if two detections in different models are the same object or not. Weightages to different models can also be provided to improve the model performance. For example, if one model has higher mAP then it can be assigned higher weights then the others so that the ensembling process will give that particular model a higher priority. This process will result in a better performance and higher detection rate for an autonomous system.

ii) Stage 2 Ensembling

For the stage 2 ensembling model, the images will be passed through a stock YOLOv4 model that detects the COCO 90 classes then further merged with output from the stage 1 ensembled model. This will result in detection for 122 classes after stage 2 ensembling stage. However, instead of a consensus method like we used in stage 1 ensemble, we will be using an affirmative method so that it considers all the detections in the stock YOLOv4 COCO model and the stage 1 ensembled model. This implies that all the detection from stock YOLOv4 and stage 1 ensembled will be considered in stage 2 ensemble process and shown in the output.

Hence, once we have completed both the stages of ensembling, the final system will detect all the road markings, pedestrians, traffic lights, traffic signs, bikers, trucks, trains, cats, dogs, cars, and other remaining classes of the COCO dataset.

4.1.1 Improved Faster RCNN Model

Faster R-CNN is a relatively new deep learning model for object detection. It was first introduced in 2015 by k He et al. Before 2015, the most popular image object detection and classification model was Fast R-CNN, because the fast R-CNN model takes the whole image and region proposals as input in its CNN architecture in one forward propagation, and therefore shortened the training time, and improves the detection accuracy. Besides, the Fast R-CNN model also saves disk space by removing the strict requirement to store feature maps. However, the Fast R-CNN algorithm has a bottleneck in its architecture, which is the selective search region proposal generating algorithm. Since this algorithm needs to generate 2,000 proposals in each image, it takes most of Fast R-CNN's training time, and it also limits the model performance of Fast R-CNN.

The Faster R-CNN model replaced the bottleneck algorithm of Fast R-CNN model by the region proposal network. The region proposal network dramatically reduces the object detection time compared to the Fast R-CNN model. For example, using the same dataset (Pascal VOC 2012), the Fast R-CNN model takes about 2.32 seconds to detect objects, while faster R-CNN takes only 0.2 second with VGG backbone, and 0.059 second with ZF backbone. Faster R-CNN model also improves the object detection mAP using the region proposal network.

How does this "magic" region proposal network work in the Faster R-CNN model? First, we pass our image into the Faster R-CNN backbone network, and generate a convolution feature map. Then, the region proposal network takes feature maps and generates the anchors. These anchors are passed into the classification layers and the regression layers, and generate outputs.

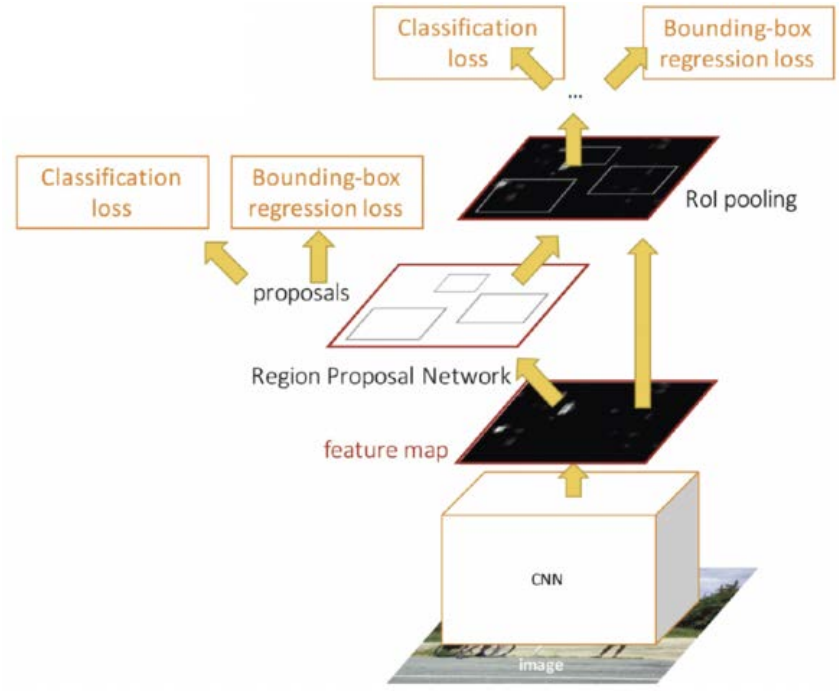


Fig 4.1.1 Network architecture of Faster R-CNN

4.1.2 Improved YOLOv4:

The You Only Look Once family of object detection models (latest version 4 was published in 2020) is a one stage algorithm that lacks the region proposal stage in algorithms such as Faster R-CNN and R-FCN. This allows YOLO algorithms to be more efficient at the expense of detection rate and accuracy. Since the YOLOv3 algorithm is a fully connected CNN, it does not use another CNN as a backbone. Instead, YOLOv3 uses a feature extractor called Darknet-53, which contains 106 convolutional layers. Low accuracy of YOLO and YOLOv2 algorithms are overcome by multi-stage object detection. Residual block in YOLOv3 helps avoid the vanishing gradient problem. A notable feature of YOLOv3 is object detection at three levels, which is achieved by down sampling layers by the factors of 8, 16, and 32. This allows YOLOv3 to detect small and large objects with better accuracy, an approach similar to that of SSD. Another feature similar to SSD is the use of 9 anchors in YOLOv3, allowing for more accurate positioning of bounding boxes.

Moreover, YOLOv3 is able to detect multiple object classes in the same image, allowing up to 10,647 bounding boxes at native input image resolution, all achieved in one pass of the fully connected convolutional layers. YOLOv3 gives up softmax activation function in object classification and adopts logistic regression and cross-entropy error measurements. This allows YOLOv3 to identify object classes that are similar or even overlapping. (Choi et al., 2019)

YOLOv4 outperforms YOLOv3 by approximately 10% in mAP and 12% in speed. The efficient model can be run on consumer grade GPU such as the GTX 1080 Ti and RTX 2070 with great accuracy and speed. The original YOLOv4 uses three backbone models: CSPResNext50, CSPDarknet53 , and EfficientNet-B3. Input network resolution is 512*512 pixels. There are between 12 and 20 million parameters depending on the backbone, neck, and head algorithms. Running on the RTX 2070 GPU, a speed of 60 FPS can be achieved with input resolution of 512*512. YOLOv4 comes with many improvements over the previous version, such as the adoption of spatial pyramid pooling (SPP) and a modified path aggregation network (PAN). The authors of YOLOv4 introduced other improvements and grouped them into two main categories: bag of freebies (BoF) and bag of specials (BoS), which refer to improvements that do not affect performance and speed, and improvements that only slightly affect performance and speed respectively. BoF includes Mosaic data augmentation, DropBlock regularization, CIoU-loss to replace Mean Squared Error loss function. BoS include Mish activation function, SPP-block and so on.

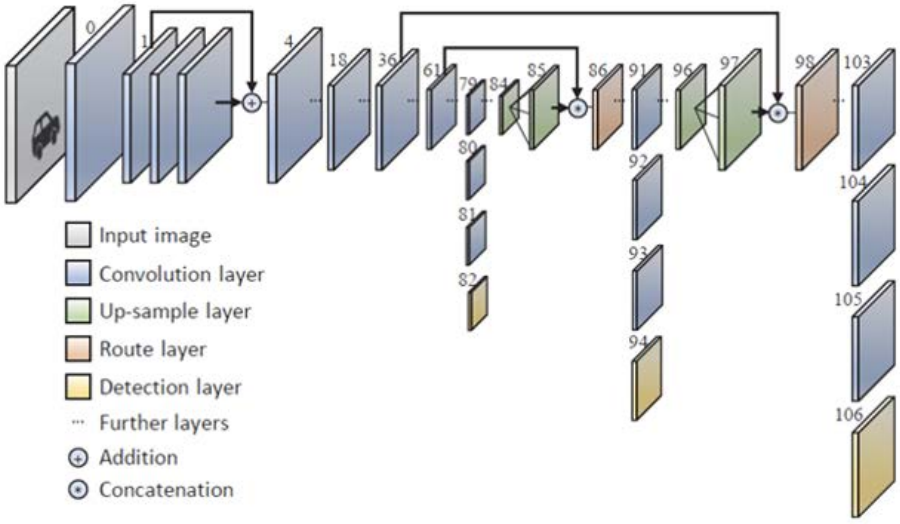


Fig 4.1.2 Network architecture of pre trained YOLOv4 model

A lot of research has been already done for traffic signs and vehicle detection. Many datasets are available which the researchers have utilized to develop many models. However, a complete road mark dataset was never found which led to very little research in its domain. Hence, for vehicles and traffic sign detection, we will be utilizing pretrained models to combine with output of road mark detection for our autonomous system. We will be implementing a pre-trained YOLOv3 for traffic sign detection and vehicle detection. For the first application, these models will be run separately to directly combine in the final output whereas in the second application, these models will be combined using transfer learning. The results for both of these applications will be compared in the model evaluation to finalize which application is better in terms of various metrics.

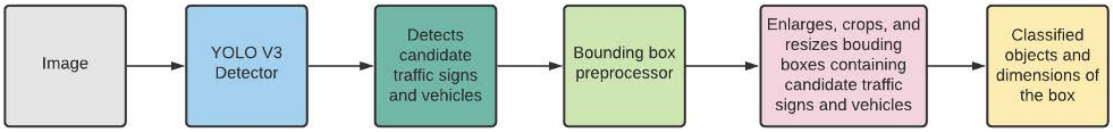


Fig 4.1.3 Stages of YOLO model

For image size 416x416 (For traffic sign images)				
Layer	Stride factor	Feature maps (SxS)	Detection objects	No of bounding boxes
82	32	13x13	Large	13x13x3=507
94	16	26x26	Medium	26x26x3=2028
106	8	52x52	Small	52x52x3=8112

Table 4.1.1 Detection of objects in YOLOv3

4.1.3 Single Shot Detector Model:

Single shot detector (SSD) is a one stage algorithm, which makes it computationally efficient. Testing speed is greatly increased, thus allowing SSD to detect and classify objects in real time. As a one stage algorithm, SSD uses convolutional layers to handle finding ROIs, give ROIs confidence ratings, classify most confident ROIs, and regressing ROI position errors all at a single pass. SSD uses VGG-16 as backbone on input data to extract features, which consists of 13 convolutional layers, 5 pooling layers, and 3 dense layers. After VGG-16, SSD uses convolution on the input image multiple times with image scaled to progressively smaller resolutions. This effectively allows SSD to create feature maps on the same image at varying scale. Larger and smaller feature maps are better at detecting smaller and larger objects, respectively. Hence SSD can achieve better detection rate and accuracy than other algorithms.

Because SSD is a one stage algorithm, an accurate position for each ROI has to be calculated in one pass, making it necessary to assign multiple bounding boxes (also known as anchor boxes in SSD) for each ground true object and then find the best fitting box for each object. SSD divides input image into user definable number of grids; when an object is found, multiple anchor boxes are placed on top of the object because the only way to determine the correct size anchor box in one pass of the convolutional network is to have multiple boxes assigned and voted upon. In terms of model structure, different size anchor boxes here correspond to different size convolutional layers, or feature maps, mentioned in the above paragraph (Jin et al., 2020).

4.1.4 EfficientDet d1 Model:

The idea behind EfficientDet arose from the effort to find solutions to improve computational efficiency by conducting a systematic study of prior state-of-the-art detection models. In general, object detectors have three main components: a *backbone* that extracts features from the given image; a *feature network* that takes multiple levels of features from the backbone as input and outputs a list of fused features that represent salient characteristics of the image; and the final *class/box network* that uses the fused features to predict the class and location of each object. By examining the design choices for these components, we identified several key optimizations to improve performance and efficiency. The well known detectors mainly rely on ResNets, RestNeXt, or AmoebaNet as backbone networks which are either not powerful enough or have problems with low efficiency.

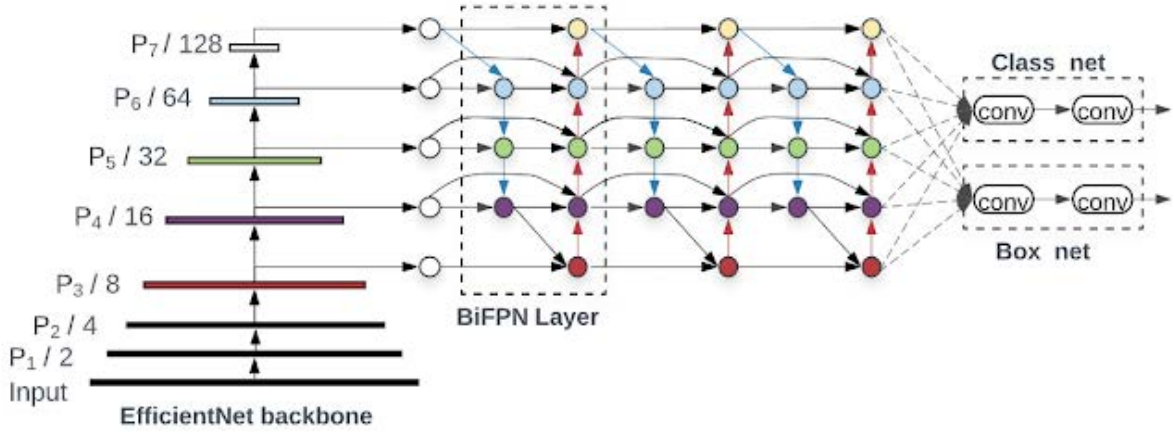


Fig 4.1.4 EfficientDet D1

4.1.5 CenterNet Model

Among deep learning models for the object detection tasks, many utilize anchors or region proposals to determine the position of objects. In order to capture accurate object positions, many models adopt the brute force approach where many bounding boxes are generated only to have most bounding boxes removed by intersection over union (IoU) and non-maximum suppression (NMS). Even though NMS is an essential process to achieve high performance, it is an undeniably costly process. To solve this problem, Duan et al. proposed a new model CenterNet with a new approach to determine the position of bounding boxes. On top of a pair of corner key points as reference, authors use an extra point at the center of the ground truth box to generate a heatmap for an object. These heatmaps determine whether a bounding box is kept or removed. Authors compared the performance of CenterNet with and without traditional NMS, and the results showed little to no improvement when NMS was applied. With

the help of the third key point and heatmap, CenterNet is able to achieve higher accuracy and speed when compared to other state of the art models. At the same inference speed, CenterNet can achieve over 40 mAP in COCO metrics, higher than YOLOv3, F-RCNN and so on. In general, CenterNet is a one stage model that is computationally efficient, can achieve high mAP and inference speed, and is not a heavy weight model in terms of resources requirement.

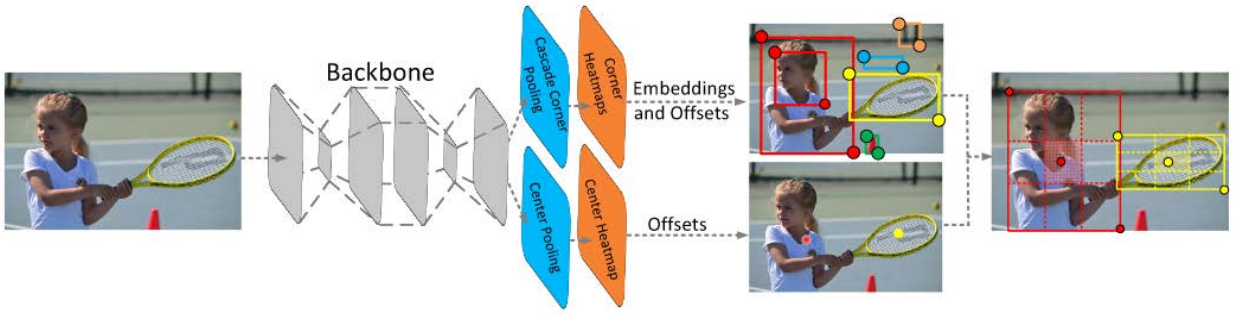


Fig. 4.1.5 CenterNet

4.2 Model Supports

The four models proposed in 4.1 above are run in the Jupyter Notebook environment using Python 3 packages such as TensorFlow and OpenCV. All model related files and code are stored in Google Drive, and the Jupyter notebooks can be run in Google Colab. Model training can be done on Colab on cloud as well as Nvidia GTX graphics processing units locally using laptops of the authors. Due to limited processing power, batch training is employed where the dataset is split into partitions and fed into the model in batches; after training each batch, the model will be saved and then retrieved for the next training.

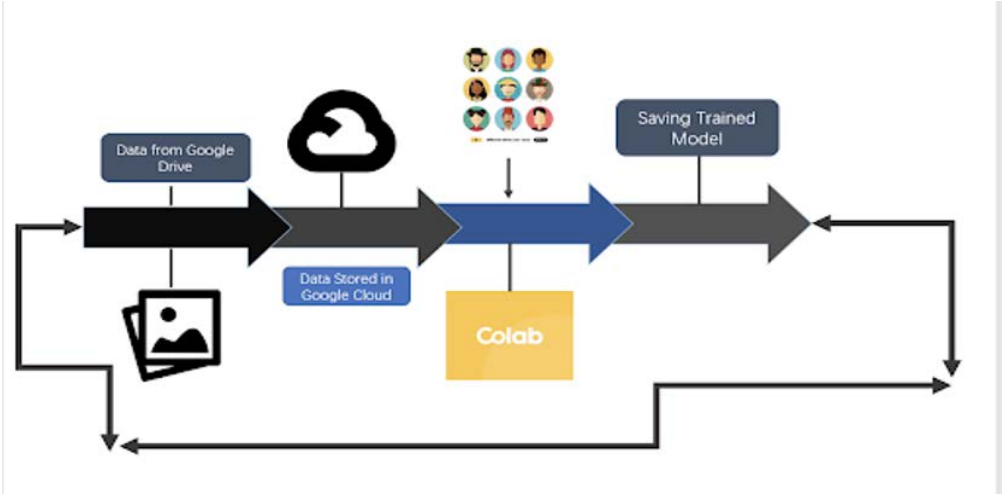


Fig 4.2.1 Data Pipeline and resource utilization for our model

Because of time and computational power constraints, transfer learning plays a big part in this research, as it dramatically shortens training time and allows authors more time to explore the topic and fine tune the models that are otherwise not feasible. The four algorithms mentioned in section 4.1 are mere "heads"; a compatible backbone is required to complete a fully functioning model. Compatible backbones include various versions of VGG, ResNet, Inception, MobileNet and so on. A head can be fit onto multiple backbones if more time and computational power is allowed to train the model from the beginning. Since authors have adopted transfer learning and intended to find pre-trained feature extractor backbones for vehicle, and road sign data, which backbone to pair with which head largely depends on the availability of pre-trained models on the internet. It will be finalized in the future during the model training phase.

4.3 Model Comparison for road mark detection and justification

The five models introduced in above section 4.1, are briefly compared in the table below. These four models are selected because they are state of the art within their family, and they hit a good balance between speed and accuracy. For object detection and tracking applied to autonomous driving, the ability to process input video footage in real time is very important; arguably as important as detection and classification accuracy. Therefore, testing speed is one of the first criteria when selecting algorithms.

The EfficientDet family of models(from D0 to D7) is under the same concepts of model scaling which is designed for different resource constraints. The performance improvement from model scaling is heavily dependent on the baseline network that the EfficientDet used neural architecture search(NAS) to find an optimum baseline network that applied reinforcement learning to help determine optimum structures with a given object function. But, the latest results from current well-written papers shows that YOLOv4 runs twice faster than EfficientDet with comparable performance. Another model that we are training is CenterNet. CenterNet is also a single shot detector algorithm and has its own advantages. It is simple and does not require Non maximum suppression or anchors. It has achieved a good performance and very good balance between accuracy and speed. It is also considered highly extensible and it can be customized in different ways. It has been so much successful that its application has been extended to various use cases apart from object detection.

Faster R-CNN in general represents the fastest testing time and frames per second (FPS) in the R-CNN family of algorithms. It has overcome many shortcomings of previous iterations of R-CNN, such as inefficient training and small object detection rate. Faster R-CNN is a two stage algorithm and inherently not as fast as one stage counterparts, but it is selected because it generally outperforms one stage algorithms in detection rate and classification accuracy. For this

reason, YOLOv4 and SSD are also included to allow comparison of speed and accuracy in empirical settings. YOLOv4 and SSD in general can process video in real time at much higher FPS. However, speed and accuracy will vary when different feature extractor backbones are applied to these four algorithms, somewhat leveling the playing field.

At this stage it is too early to determine which model is overall best. Later when we have trained all four models, we will decide which model represents the best compromise between resource requirements, accuracy, and speed. For now we know that Faster-RCNN represents the highest accuracy due to its two stage architecture; SSD represents the fastest among one stage models; and YOLO strikes the best balance. Before we have trained the models, we can only refer to existing publications to predict accuracy and speed. Below is a table comparing predicted accuracy and speed of the four models.

Model	Backbone	Unique Feature Extraction	Predicted Precision mAP	Predicted Speed/FPS (varies depending on different hardware and input size)
Faster RCNN	VGG 16, Inception v2 ResNet-50 ResNet-101 Inception ResNet v2	Region Proposal Network (RPN) which directly generate region proposals, predict bounding boxes and detect objects	80%	18
SSD	MobileNet	Fully connected convolution for bounding box location and object class.	46%	13

EfficientDet1	ImageNet	Scalable and efficient object detector which can adapt to a wide range of resource constraints	80%	1
YOLOv4	CSPResNext50, CSPDarknet53	Bag of Freebies and Bag of Specials object detection. Uses CIoU loss instead of MSE. Mish activation function.	78%	20

Table 4.3.1 Comparison of object detection models

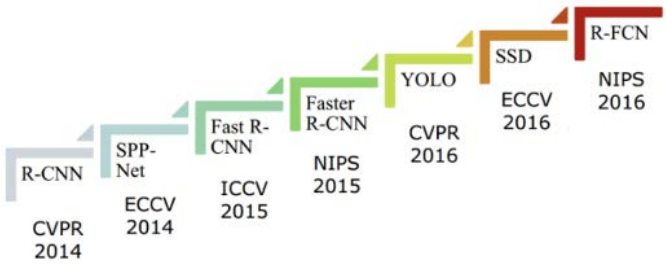


Fig 4.3.1 Timeline of object detection models

4.4 Model Evaluation Methods

An object detection model first detects by predicting the coordinates and creating bounding boxes around the objects which is called localization, and then classifies the detected objects. To evaluate these models, conventional evaluation methods do not show accurate results. Here, the performance of both localization and classification needs to be evaluated with data from created bounding boxes and coordinates in the XY plane. Hence, we will be using multiple methods to determine the evaluation metrics for our model and each of the classes in our model. Mean Average Precision, Intersection Over Union, Precision rate, Recall rate, and F1 score are most commonly used as evaluation metrics for an object detection model.

Intersection Over Union (IoU):

IoU takes the actual bounding box from the labelled data and the detected bounding box given by the model and then it calculates the intersection over union of these bounding boxes to evaluate the resultant detection. IoU of 1 implies that the detected bounding box area is the same as the ground truth mentioned in the label. A threshold value can be set to select detections only above a certain value and filter out the rest.

$$IoU = \text{Intersection Area} / \text{Union Area}$$

Confusion Matrix:

For our object detection model, we have calculated the True Positive, False Positive, False Negative which will let us create a confusion matrix and evaluate metrics like precision, recall, Mean Average Precision.

		Actual class	
		positive class	negative class
Predicted class	positive class	True Positive(TP)	False Positive(FP)
	negative class	False Negative(FN)	True Negative(TN)

Fig. 4.4.1 Confusion Matrix

TP	Predicted positive and it is true.
TN	Predicted negative and it is true.
FP (type I error)	Predicted positive and it is false.
FN (type II error)	Predicted negative and it is false.

Confusion matrix summarizes most of the commonly used evaluation metrics such as precision, recall, and F1-score. For our object detection model, we will calculate these metrics for the whole model as well as for each of the classes to demonstrate the results. Precision and recall can be calculated as:

Precision = TP/(TP+FP) where TP is True Positive amd FP is False Negative

Recall = TP/(TP+FN) where FN is False Positive

Similarly, the F1 score is the harmonic mean of the precision and recall metrics. Blending the two is useful; precision measures how effectively the classifier performs when it is predicting a '1', whereas recall measures how many of the total '1' classes out of all the 1-labeled observations were predicted correctly. F1 will be calculated as:

F1 = 2*precision*recall/(precision+recall)

Average Precision (AP):

The average precision summarizes the precision and recall curve by representing as a single value known as the average of all precisions. It is calculated by taking a loop for every precision-recall and the difference in next and current recall is multiplied by the current precision. AP can be also defined as the weighted sum of precisions at each threshold where the weight is the increase in recall.

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k)$$

$$Recalls(n) = 0, Precisions(n) = 1$$

$$n = Number\ of\ thresholds.$$

Mean Average Precision (mAP):

The mean average precision is the average of APs of all the classes.

$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k$$

$$AP_k = the\ AP\ of\ class\ k$$

$$n = the\ number\ of\ classes$$

4.5 Model Evaluation and Validation at Training Phase:

We are training five different models to detect all the classes and then ensemble the output of some models which will give us better results. As described in section 4.2, the five models used are YOLOv4, SSD, Faster RCNN, CenterNet, and EfficientDet1. The result of YOLOv4 trained on all the 32 classes from roadmarks, traffic signs, vehicles, and pedestrians is explained below.

4.5.1 YOLOv4 Evaluation

The YOLOv4 model was trained on 10,544 images. These images were further split in 80:20 ratio for training and validation. The model was trained for 5200 iterations and with batch size set to 32. The subdivision was set to 16 which results in minibatch size equals to 2. As shown in the figure below, the loss rate went under 2 and the average loss was 2.0112 for the final iterations. The loss decreased drastically in the beginning as expected and was almost constant when reaching around 2. As Mean Average Precision is the one of the important evaluators, the model was evaluated for mAP @ 0.50 and then further evaluated for Intersection over union (IoU). The mAP@0.50 was 74.5% and average IoU was found to be 57.24%. The confidence threshold was taken as 0.25 to calculate the above metrics.

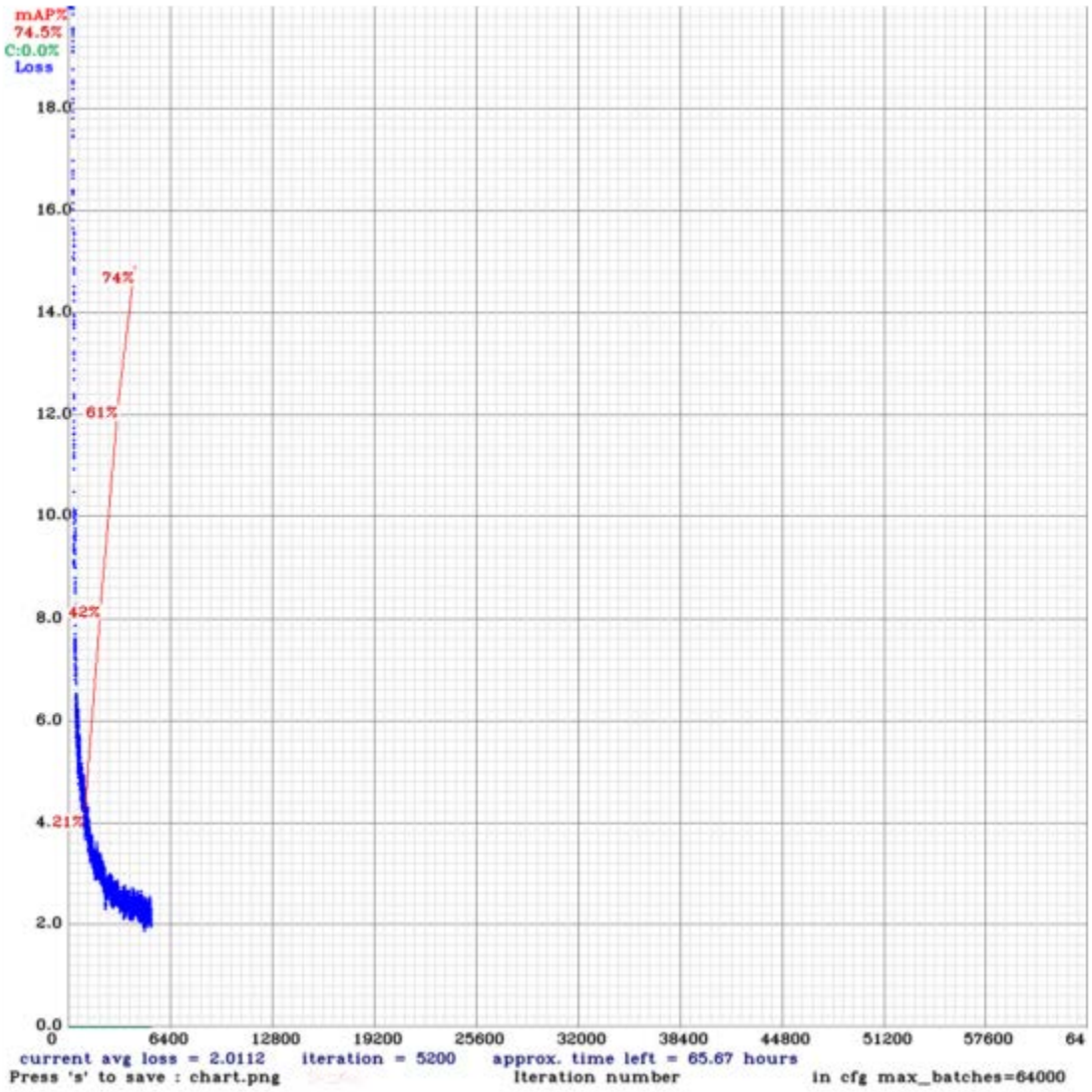


Fig. 4.5.1 Loss vs Iterations

Apart from the metrics explained above, the model was also evaluated for precision, recall, and F1-score. The average precision for the model at 0.25 confidence threshold was found to be 0.79 and average recall was 0.57 for the model. The true positive (TP) was found to be 2560, false positive (FP) as 668, and false negative as 1911. Hence, the precision for the model turns out to be = 2560/(2560+668) = 0.79. Similarly, recall = 2560/(2560+1911) = 0.5725

All the metrics shown above are for the whole model. The average precision was also calculated for each of the classes in the model. The following table shows the average precision for each of the classes.

Class ID	**Class**	**Average Precision (%)**	**Class ID**	**Class**	**Average Precision (%)**
0	25	86.54	16	Rail	98.41
1	30	94.73	17	Right	89.46
2	35	97.38	18	School	52.03
3	40	99.12	19	Signal	88.01
4	45	99.85	20	Stop	86.76
5	50	92.14	21	Crossing	85.74
6	Bike	84.25	22	Yield	81.17
7	Bus	85.32	23	Car	90.42
8	Diamond	86.03	24	Motorcycle	80.60
9	Forward	77.99	25	Person	85.91
10	Forward Left	95.49	26	Truck	72.07
11	Forward Right	93.14	27	Traffic light	76.13
12	Keep Clear	87.21	28	Pedestrian Sign	96.57
13	Forward Left Right	93.78	29	School Sign	78.80

14	Left	86.69	30	Yield Sign	77.40
15	Pedestrian	97.27	31	Stop Sign	83.50

Table 4.5.1 mAP for YOLOv4

The YOLOv4 model was able to accurately detect most of the classes every time on the test data. Apart from the unseen test images, the model was also tested on multiple videos to validate the evaluation results given in section 4.5.1. Some of the test results for different classes are shown in the images below

Fig 4.5.1 STOP with 95 % confidence

Fig 4.5.2 Speed Limit 25 in opposite directions with 84% and 75% confidence

Apart from the images in the test dataset and unseen images from the internet, the model was also tested on multiple videos from San Jose downtown. In the video, the model was able to identify all the road marks very accurately. However, it struggled to identify classes of the pretrained models such as cars, and traffic lights. Some of the frames of detection on videos are shown below:

Fig 4.5.3 Pedestrian (56% confidence) and Crossing (32% confidence)

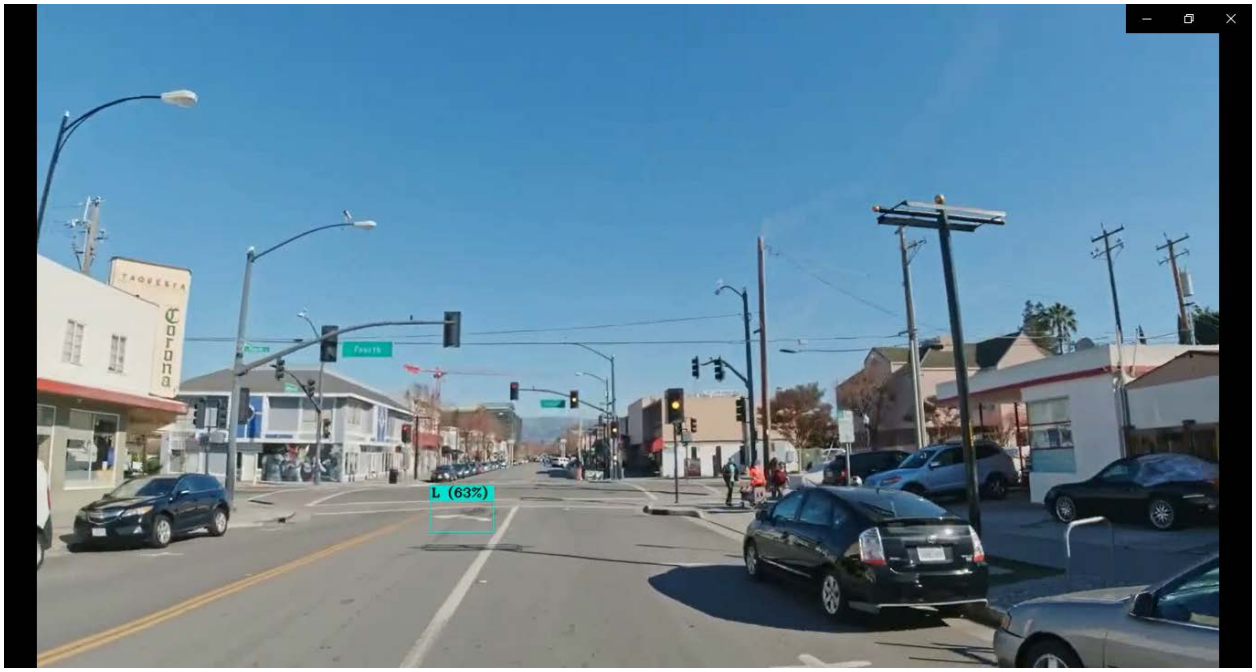


Fig 4.5.4 Left (63% confidence)

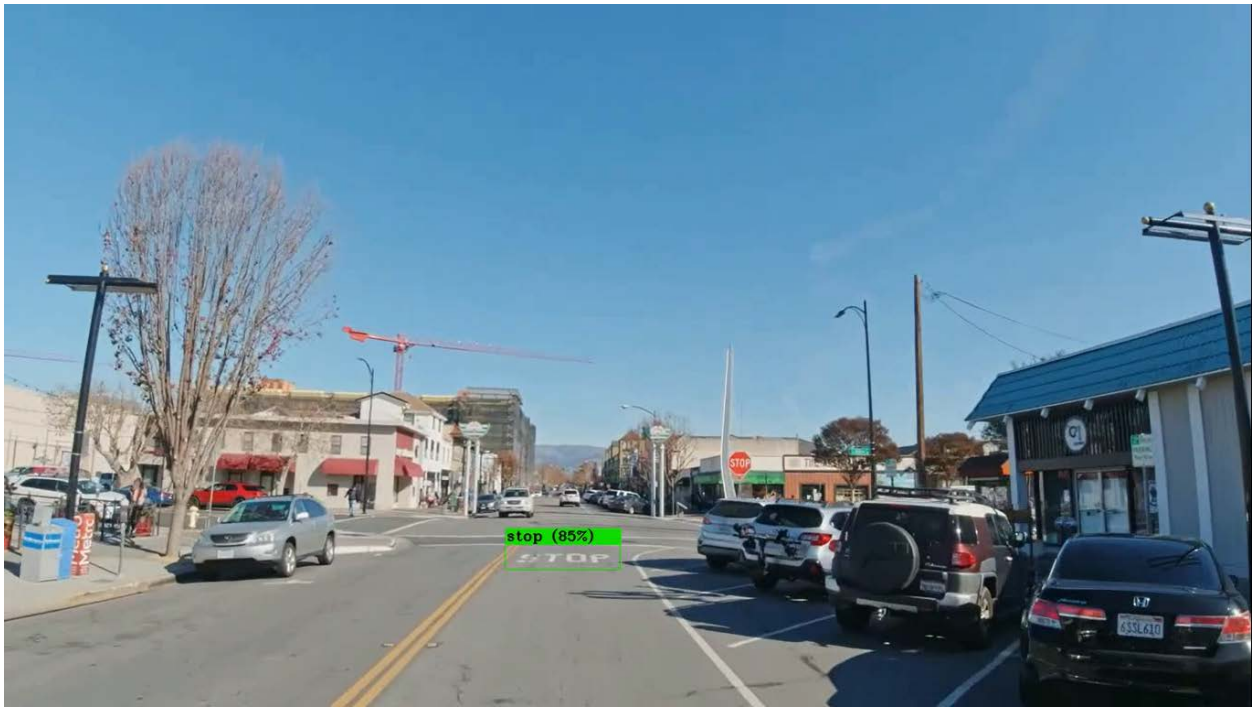


Fig 4.5.3 STOP (85% confidence)

These detections from the YOLOv4 model will be ensembled along with the detections of the other three models that are currently being tested. The objects detected in at least two models will be selected and shown in the output as bounding boxes.

4.5.2 Faster R-CNN Evaluation

Our Faster R-CNN model uses the ResNet 101 backbone, and has 4 sequential layers in total. Each of the 4 sequential layers are composed of convolutional layers, batchnorm layers and relu layer. Faster R-CNN model provides a good solution to our targeted problems from the following perspectives:

1. Faster Training Time:

Thanks to the Region Proposal Network architecture of Faster R-CNN, this model takes only about 16 seconds for every 100 iterations. Therefore, in a given limited time, the Faster R-CNN model can be trained for 14 epochs with 12,726 iterations in each of the epochs, and reaches its best model performance for traffic signs and road markings detection.

```
[session 1][epoch 14][iter 5000/12726] loss: 0.1442, lr: 1.00e-04
                        fg/bg=(32/96), time cost: 15.982504
                        rpn_cls: 0.0092, rpn_box: 0.0125, rcnn_cls: 0.0253, rcnn_box 0.0533
[session 1][epoch 14][iter 5100/12726] loss: 0.1608, lr: 1.00e-04
                        fg/bg=(32/96), time cost: 16.027322
                        rpn_cls: 0.0244, rpn_box: 0.0928, rcnn_cls: 0.0704, rcnn_box 0.1429
[session 1][epoch 14][iter 5200/12726] loss: 0.1749, lr: 1.00e-04
                        fg/bg=(32/96), time cost: 16.052711
                        rpn_cls: 0.0070, rpn_box: 0.0687, rcnn_cls: 0.0426, rcnn_box 0.0652
[session 1][epoch 14][iter 5300/12726] loss: 0.1518, lr: 1.00e-04
                        fg/bg=(32/96), time cost: 16.191949
                        rpn_cls: 0.0041, rpn_box: 0.0180, rcnn_cls: 0.0175, rcnn_box 0.0282
[session 1][epoch 14][iter 5400/12726] loss: 0.1798, lr: 1.00e-04
                        fg/bg=(32/96), time cost: 16.257116
                        rpn_cls: 0.0069, rpn_box: 0.0173, rcnn_cls: 0.0251, rcnn_box 0.1082
[session 1][epoch 14][iter 5500/12726] loss: 0.1733, lr: 1.00e-04
                        fg/bg=(32/96), time cost: 16.172179
                        rpn_cls: 0.0416, rpn_box: 0.0769, rcnn_cls: 0.0471, rcnn_box 0.1475
[session 1][epoch 14][iter 5600/12726] loss: 0.1768, lr: 1.00e-04
                        fg/bg=(27/101), time cost: 16.069873
                        rpn_cls: 0.0174, rpn_box: 0.0174, rcnn_cls: 0.0082, rcnn_box 0.0322
```

Fig. 4.5.4 Iterations of Faster RCNN

2. Relatively High mAP:

After training for 14 epochs with 12,726 iterations per epoch, the trained Faster R-CNN model was tested on testing images from different sources with different resolution, such as web images, Google Street View images, images captured from video, etc.

Below is the testing result for each class of our project:

Classes	**mAP**	**Classes**	**mAP**	**Classes**	**mAP**	**Classes**	**mAP**
25	0.7922	**Bus**	0.8298	**L**	0.7014	**Xing**	0.8624

30	0.7842	**Diamond**	0.7192	**Pedestrian**	0.4306	**Yield**	0.8556
35	0.9033	**F**	0.7086	**Rail**	0.9015	**Car**	0.7786
40	0.9052	**FL**	0.6051	**R**	0.5096	**Biker**	0.5343
45	0.8884	**FR**	0.6059	**School**	0.7995	**Traffic Light**	0.5268
50	0.9869	**KC**	0.5024	**Signal**	0.8431	**Yield Sign**	0.5247
Bike	0.7791	**FLR**	0.6055	**Stop**	0.8833	**Truck**	0.7093
School_Sign	0.6489	**Stop_Sign**	0.8709	**Ped_Sign**	0.7116	**Ped**	0.9071

Table 4.5.2 Faster RCNN mAP

3. Capability of Real-Time Object Detection

Due to the rapid processing time of the region proposal network architecture, our trained Faster R-CNN model is capable of detecting trained road marks and traffic sign classes, pedestrian, truck, cars, bike, and traffic lights in real-time video. Our trained Faster R-CNN model is able to connect to web cameras, and detect those classes of object in real-time.

Faster R-CNN Model Parameters:

Model Backbone	**#GPUs**	**Batch Size**	**Learning Rate**	**Learning Rate Decay**	**Max Epoch**	**Time/Epoch**	**Memory/GPU**
ResNet 101	2	1	1 e-3	10	14	1.48 hr	5291 MB

Table 4.5.3 Faster RCNN Parameters

1. mAP (see above chart for detail mAP of each class)
2. Loss Curve:

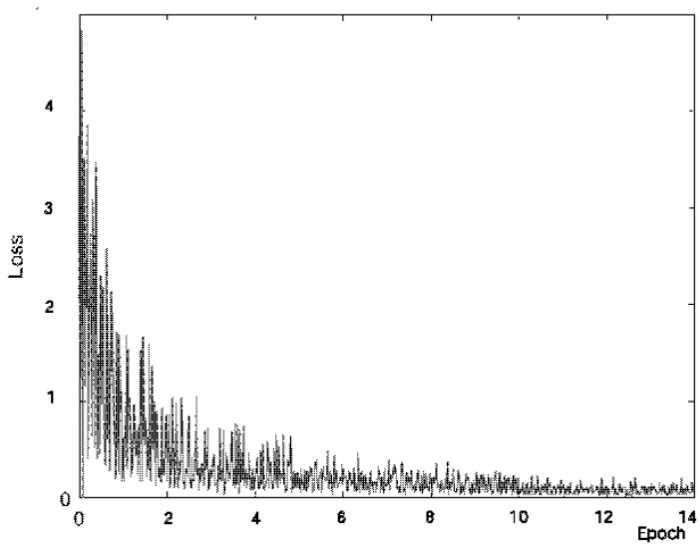


Fig. 4.5.4 Faster - RCNN Loss Curve

3. Road Marking and Traffic Sign Recognition Result

Below are the Faster R-CNN model detection results on images with different resolution from different sources. Sources of test images including google street view images, google satellite dashcam videos (day and night view), web crawling images, etc.

Fig 4.5.5 Google Street View Images Detection Result (Faster R-CNN)

Fig 4.5.6 Dashcam Detection Result (Faster R-CNN)

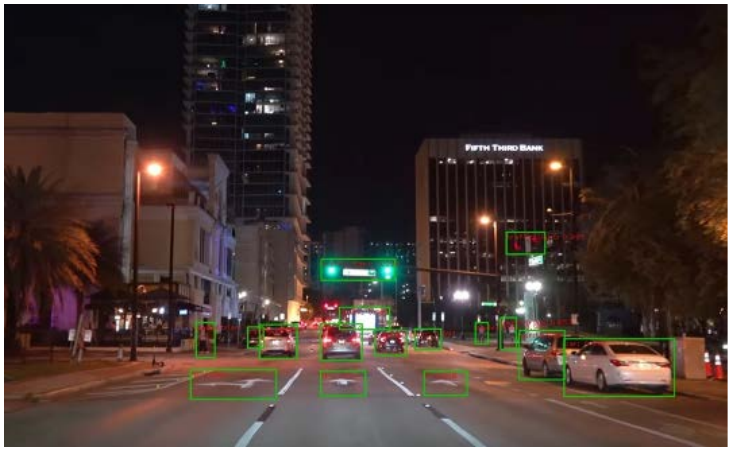


Fig 4.5.7 Dashcam Night View Detection Result (Faster R-CNN)

4.5.3 SSD Model Evaluation

In later sections of this paper, the concept of mean average precision (mAP) will be covered in more detail. In this section, mAP is a measure of model prediction accuracy. As mentioned in the above section, our custom YOLOv4 model achieves 0.72 mAP@0.5. For our SSD model, mAP@0.5 is 0.69 for all object sizes and a maximum of 100 detections per image.

Below is a screenshot of our early SSD model evaluation in COCO metrics. This model uses MobileNetV2 FPN with input size of 320x320; model is trained with 23 classes of road markings, which is not the final integrated 32 classes including cars, pedestrians etc. Reason for distinguishing between 23 road marking classes and 32 comprehensive classes will be covered with more details in later sections; in summary, models trained with road markings only will not detect cars, pedestrians etc. That is why we had to add additional object classes and re-do the annotations for the original images to include the additional classes.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.375
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.694
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.373
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.084
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.379
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.467
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.366
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.521
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.525
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.215
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.512
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.632
```

Figure 4.5.3.1 Early stage SSD model mAP table

For models built on Tensorflow framework, it is possible to load TensorBoard to track the training metrics including loss for classification and localization, total loss, regularization loss, and custom learning rate. Below is a screenshot of a sample Tensorboard training metrics of our SSD model. Again this particular model is trained with 23 classes of road markings.

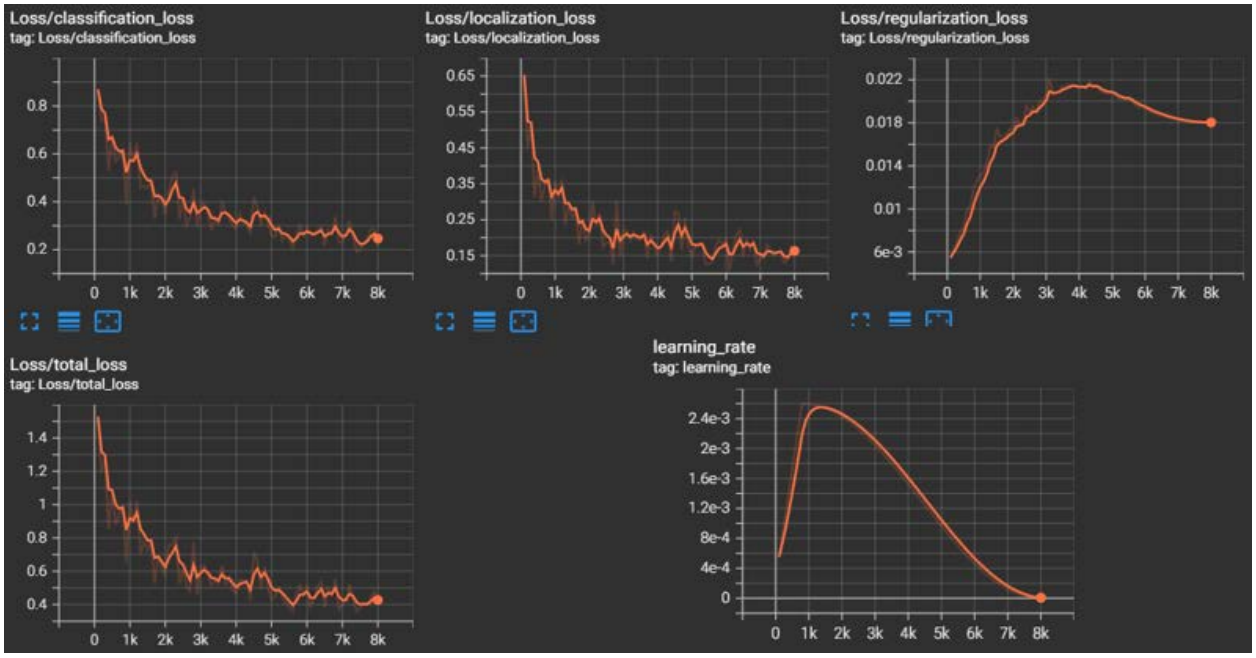


Figure 4.5.3.2 Early stage SSD model training loss and learning rate

After we went back to the dataset and added annotations for 9 more classes, the SSD model is trained with the updated dataset from scratch. The SSD model also has an upgraded input size from 320x320 to 640x640 pixels. The increase in input training size should help with detection accuracy especially on smaller objects. In a bigger image, a small object now occupies more pixels and is more likely to be detected. As with any increase in model size, training time and required hardware resources are greatly increased. The same RTX 3060 laptop GPU with 6GB of vram can only fit a batch size of 5, down from 20+ in the original SSD model. Below is a screenshot of the final SSD model training evaluation.

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.227
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.439
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.207
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.079
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.272
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.250
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.294
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.429
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.433
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.155
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.478
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.542
INFO:tensorflow:Eval metrics at step 20000
```

Figure 4.5.3.3 Final SSD model mAP metrics

4.6 Model Ensemble

Model ensembling or stacking is a common strategy when dealing with resource constraints. While the former helps solve the problem of limited hardware resources we have, we rely on ensembling to expand the functionality of our models. Hence we propose a two stage ensemble for consolidating model predictions and expanding functionality. Thanks to the open source program Ensemble Boxes, we are able to utilize a novel approach Weighted Boxes Fusion (WBF) to perform ensembling.

Our two stage ensemble involves a YOLOv4 in its stock form trained on COCO dataset, as well as all four of our custom trained models, including CenterNet, EfficientDet d1, SSD, and YOLOv4. Our custom trained Faster R-CNN achieves respectable accuracy, but speed and resources requirement is not on the same level as the other models. For this reason, Faster R-CNN is not included in our two stage ensemble.

In Stage 1 of the ensemble, all four of our models are tested on the same dataset. Then all predictions are gathered and passed into ensemble using WBF to generate one set of combined bounding boxes for each image. In Stage 1, we use the consensus method for ensembling as we aim to improve the detection accuracy without generating redundant predictions. WBF is superior to simple NMS and other traditional methods of ensembling; all bounding boxes are

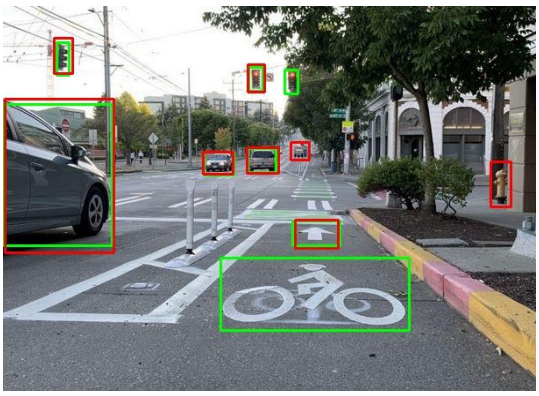
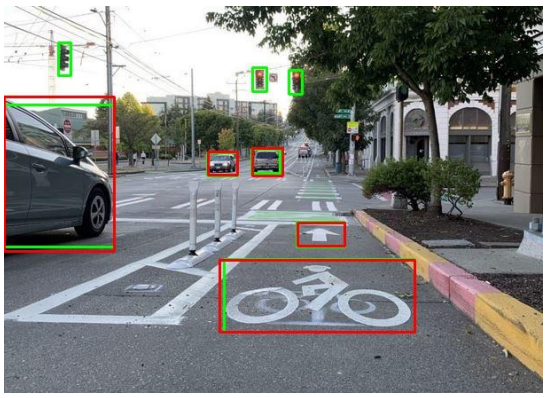
considered and a new bounding box is created by averaging the positions of all predicted bounding boxes for the same object. Therefore, WBF generally achieves higher precision and better bounding boxes with higher IoU. (Solovyev et al., 2021)

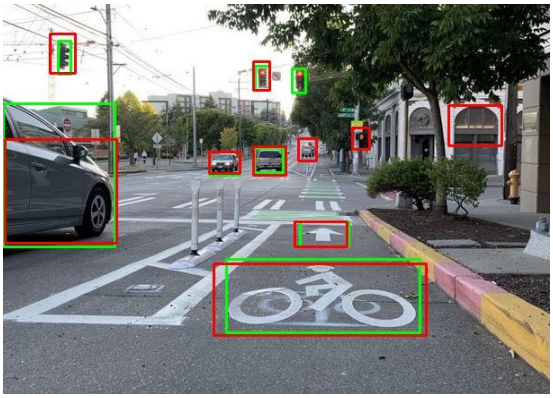
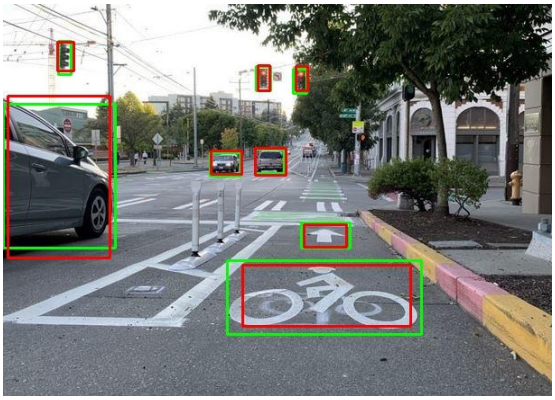
In Stage 2 of the ensemble, we take the Stage 1 output and output from a stock YOLOv4 trained on COCO dataset to perform another round of ensemble. In Stage 2 we use the affirmative method to ensemble, essentially combining the improved predictions of road markings from Stage 1 ensemble and the ability to predict 80 COCO classes from a stock YOLOv4 model. However, our dataset was not annotated for all 80 COCO classes, and so we are not able to calculate mAP of Stage 2 output.

Below is the mAP of Stage 1 ensemble output, and some samples comparing the same test image of different models including Stage 1 ensemble.

COCO METRICS:
AP: 0.3872353337220656
AP50: 0.7279107534622377
AP75: 0.36487266312737554
APsmall: 0.17347364458807407
APmedium: 0.41712147490052354
APlarge: 0.4629853759649412
AR1: 0.323351944045684
AR10: 0.4458371575797668
AR100: 0.44591120468926215
ARsmall: 0.19912293679418158
ARmedium: 0.4815303470881397
ARlarge: 0.5122397981365311

Fig 4.6.1 Stage 1 Ensemble mAP

a)  b) 

c)  d) 

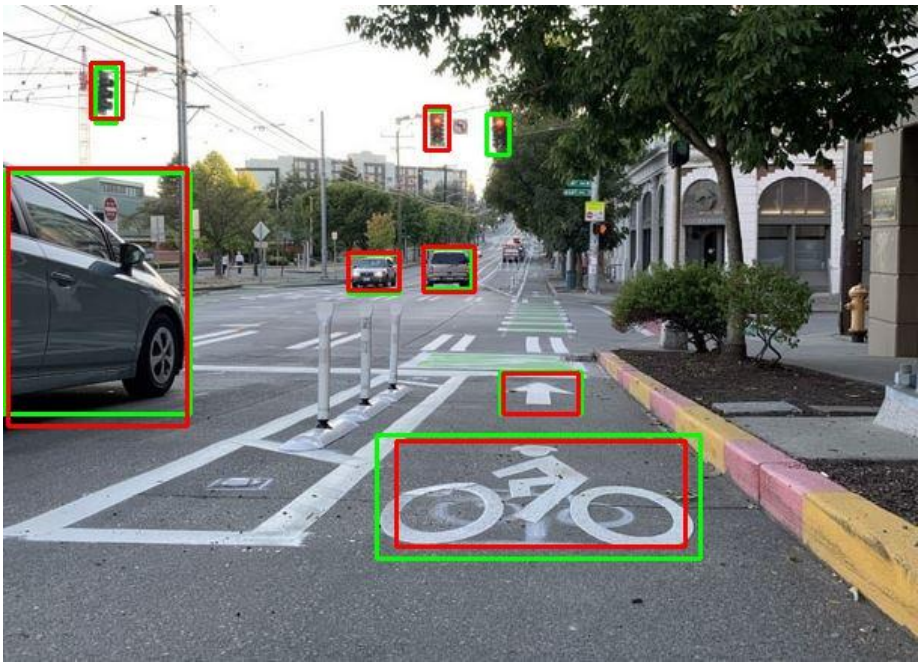
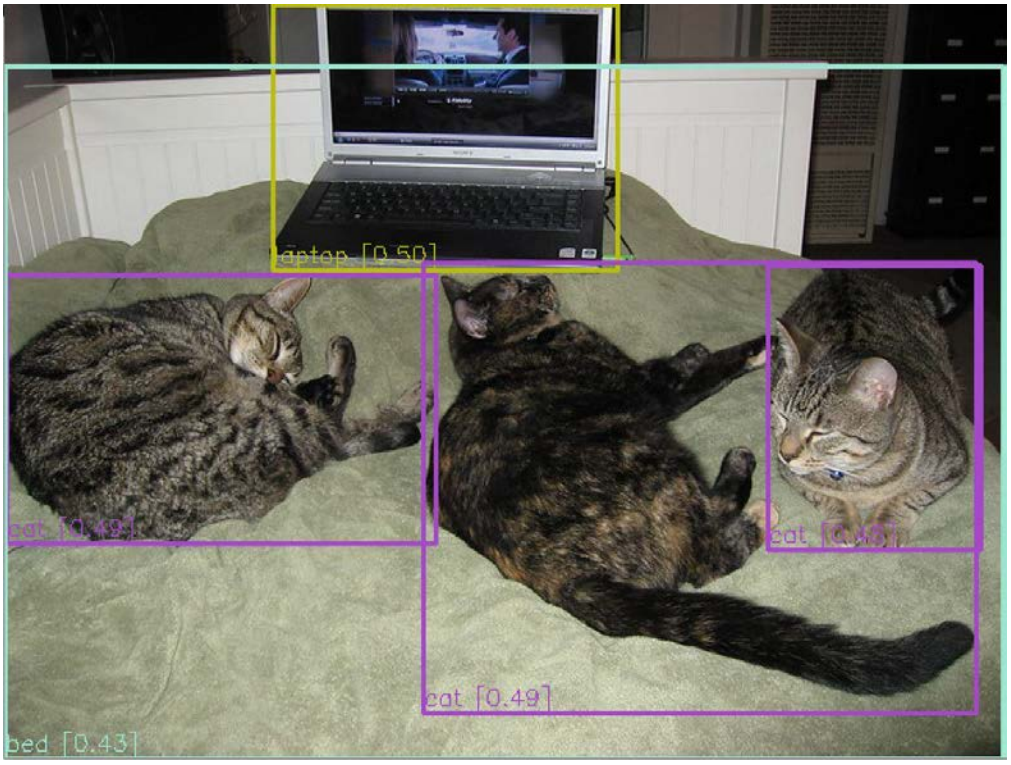
e) 

Fig 4.6.2 a) CenterNet prediction, b) EfficientDet prediction, c) SSD prediction, d) YOLOv4 prediction, e) Stage 1 Ensemble prediction

For Stage 2 Ensemble, because our goal is to expand functionality in detection, we will focus on visualizing the ensemble output in the 6 images below. In these images, the Stage 2 Ensemble model is able to detect many objects other than road markings, all while maintaining excellent accuracy in detecting road markings as shown in the last image.


laptop [0.50]
cat [0.49]
cat [0.49]
bed [0.43]


person [0.43]
dog [0.49]
bicycle [0.41]


traffic_light [0.47]
traffic_light [0.71]

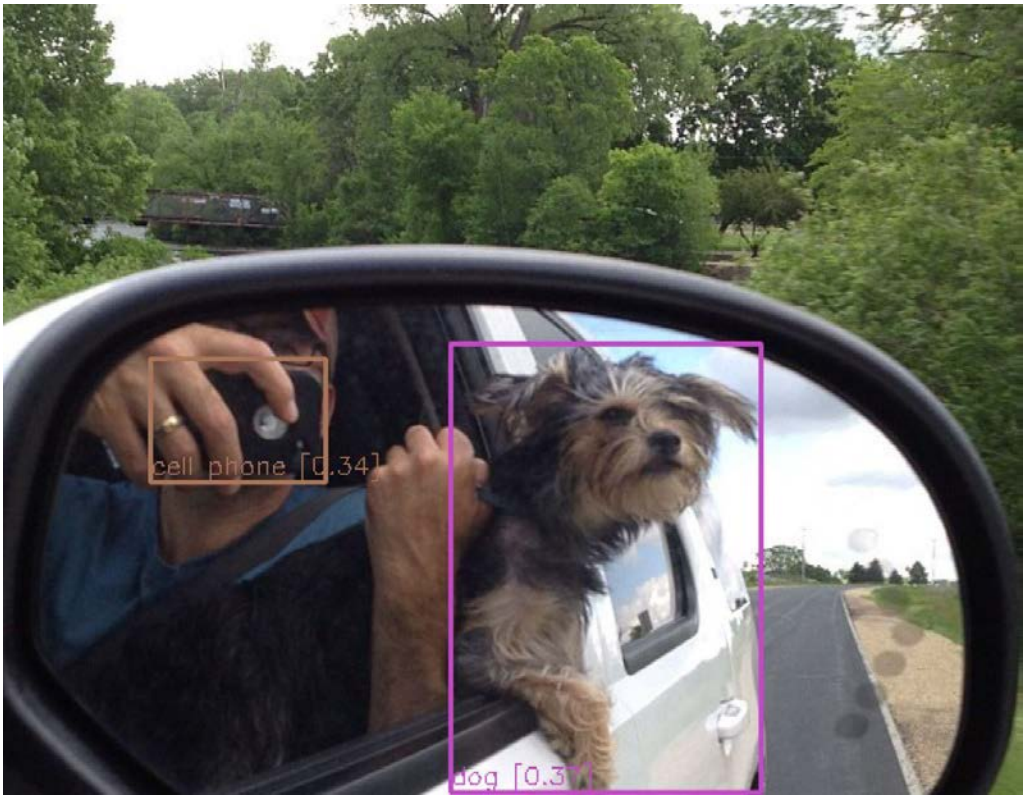

cell phone [0.34]

Figure 4.6.3 Results of Stage 2 Ensemble model in 6 images

5. Data Analytics and Intelligent System

5.1 System Requirements Analysis

The goal of this project is to create an object detection system that can be integrated into the autonomous driving system of vehicles. And this system is designed to have access to the autonomous vehicles' (AV) main computing unit, storage, video capturing devices, and various displays and monitors. Actors of the system include drivers and system engineers. While AV system engineers are able to access all components of the object detection system, AV drivers only have partial access, namely the final output of the fully integrated AV system outputs on the main display.

The object detection system is designed for five use cases under two scenarios. For real world scenario:

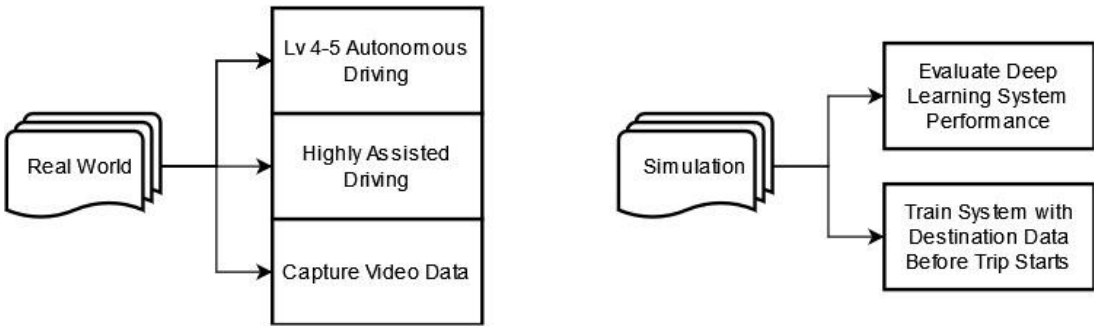
- Fully autonomous driving; SAE Level 4 high automation and Level 5 full automation. AV system takes into account the output of the object detection system and takes full control of the vehicle.
- Highly assisted driving. AV systems display the output of the object detection system to the driver and give warnings in potentially dangerous situations.
- AV captures and stores videos of local road conditions, regional road signs etc, and engineers can use this data to retrain the machine learning component.

For simulated scenario:

- System overall performance and/or deep learning components can be evaluated by testing with images, videos or simulated driving situations.
- AV systems can achieve higher performance that is tailored for the destination city, by training deep learning models with destination images and videos beforehand.

The first two use cases are an enhancement of existing AV systems. With this object detection system and road markings database, AVs will be able to identify major U.S. road markings and use this information to either make decisions of vehicle movements or pass the information to the driver. The third use case is an expansion of existing AV systems. The machine learning component of this system can be utilized to capture new image and video data and ultimately help improve model output accuracy. By capturing videos of local driving conditions, the system is able to detect and classify objects more accurately. The system can also gain the ability to detect new or unique signs in the local area where the AV usually operates at. The third use case allows AVs to continually improve the autonomous driving function, given that the AV manufacturer assigns engineers to maintain the system and retrain the machine learning component frequently. The fourth use case allows AV system engineers to evaluate new models or model revisions before deployment. Engineers can also finetune the system to tailor fit different AV equipped with different image capturing systems. Last but not least, the fifth use case can greatly enhance the performance of the object detection system in the destination city, by feeding and training the deep learning components with destination city data before the trip.

As mentioned, this system is designed to be integrated into the AV system. Therefore, this system needs to be lightweight enough to be stored in the AV main storage and memory, and the inference component needs to be efficient for existing AV hardware to execute without exhausting processing power. Most object detection deep learning models take less than 200MB in size, and we anticipate our final system to be within the 1GB mark. Fast solid state storage has become economical and widely available, and a 1GB storage requirement is minimal in today's standard. Inferencing is no longer a computational heavy task with today's hardware. We expect future AVs to have computing units that can handle our system relatively easily. For example, Tesla's Hardware 2 with Nvidia CUDA support should be adequate.



5.2 System Design

While this system is designed to be integrated into AV main systems, it is not designed with proprietary code in mind, but to be universal and compatible with all AV manufacturers given some minor adjustments. Thus the base code is written in Python language to maximize compatibility in different systems. To be more specific, the deep learning models are in Python format .py and Jupyter Notebook format .pynb. Training and inference can be run with Python, Anaconda, Jupyter, Linux, Windows, AWS Sagemaker environments. As this system is highly configurable, it can reside simultaneously on AWS and inside the AV computer. Communication between object detection components, AV systems, and environments is expected to be in the form of function calls and application programming interface (API).

Input of the system is videos captured by AV's cameras. Depending on the AV, there can be multiple cameras; which camera to choose depends on the design of the particular AV. Video then enters the AV main computer. From here on the computer will redirect the input to different components depending on the use case. For real world driving scenarios, if the AV is equipped with an adequate graphics processing unit, it can send input to deep learning components and run inference directly. If the 5G network is fast and the signal is good in the city, AV computers can stream input video to a cloud environment, such as Amazon S3 database for storage and Amazon

Sagemaker for deep learning model training and inference. Amazon Sagemaker supports many deep learning infrastructures such as Tensorflow, MXNet, PyTorch and so on; it is compatible with our object detection system. With the rapid adaptation of 5G networking technology, it is possible for AV driving at high speed to send and receive data for object detection in real-time [reference Telekom/T-mobile]. An AV computer will then integrate the object detection inference output with other autonomous driving algorithms to generate the final output. In level 4 or 5 highly autonomous driving scenarios, the final output will be responsible for operating the vehicle. In highly assisted driving scenarios, AV will display the final output on the main display for the human driver.

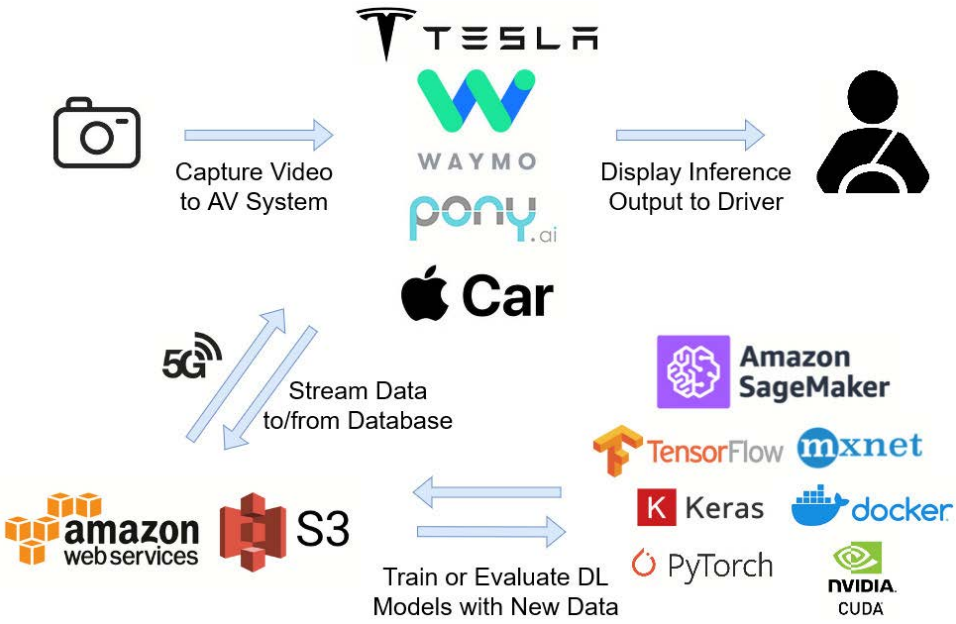


Fig 5.2.1 Potential System Design

5.3 Intelligent Solution

5.3.1 Developed AI and machine learning solutions:

The aim of the system developed is to detect all the 23 roadmarks, cars, bikers, trucks, pedestrians, traffic lights, and traffic signs making it a fully autonomous system. To detect all these classes, we developed multiple machine learning models all doing the same task to detect all the classes. As discussed before, these machine learning models are YOLOv4, SSD, Faster RCNN, and R-FCN. However, all these models were introduced at different times and have their own advantages and disadvantages. If output from only one of these models is taken then it will get the advantages of that model but also will result in performance due to limitations in the model. Hence, to overcome this, we are ensembling the bounding boxes generated by all the 4

deep learning algorithms and improve the accuracy, intersection over union, and mean average precision.

Given the unique capabilities of AVs, we want to ensure that their driving patterns are designed for maximum impact on roadways. The proper deployment of AVs should minimize gridlock, decrease total energy consumption, and maximize the capacity of our roadways. While there have been decades of research on these questions, there isn't an existing consensus on the optimal driving strategies to employ, nor easy metrics by which a self-driving car company could assess a driving strategy and then choose to implement it in their own vehicles. With properly designed benchmarks we can examine an AV's driving behavior and quickly assign it a score, ensuring that the best AV designs are the ones to make it out onto the roadways. Furthermore, benchmarks should facilitate research, by making it easy for researchers to rapidly try out new techniques and algorithms and see how they do at resolving similar problems.

The system is designed to work in both the simulation and real environment. The potential use case of the system could be applied in multiple dominos. The module is designed to face the real world autonomous driving situation. To be more specific, we need to understand the advances of deep architectures for lane marking detection are inseparable from that of semantic segmentation network structures. The problem can be framed as a classification study in which the system can output the predicted labels which correspond to different road marks. As we mentioned before, the three models we used (Yolo v3, F-RCNN and SSD) were trained and will integrate to the system as a comprehensive ensemble model in the end.

5.3.2 Input and Output Requirements, Supporting Systems and Solution APIs

The input dataset for training in this project was the roadmark dataset that we collected using Restful APIs from Google Street View and maps and the images collected manually using dashboard on the car and cropping existing images from the internet. The udacity dataset was also used for other classes to train the machine learning models. Once our system is developed, to predict, the input to our system can be any image, a series of images, or a video of a road, sidewalk, pavement, or objects like cars, and pedestrians in any environment. As the ultimate aim of the system is to detect the objects, the input feed will be processed according to the series of steps that we have aligned for detection. These steps include converting the images into a specified dimension first, pass through all the 4 models, generate bounding boxes, convert bounding boxes into JSON file, fetching box dimensions along with detection labels and confidence score which is further transferred to the final ensembling step of the system.

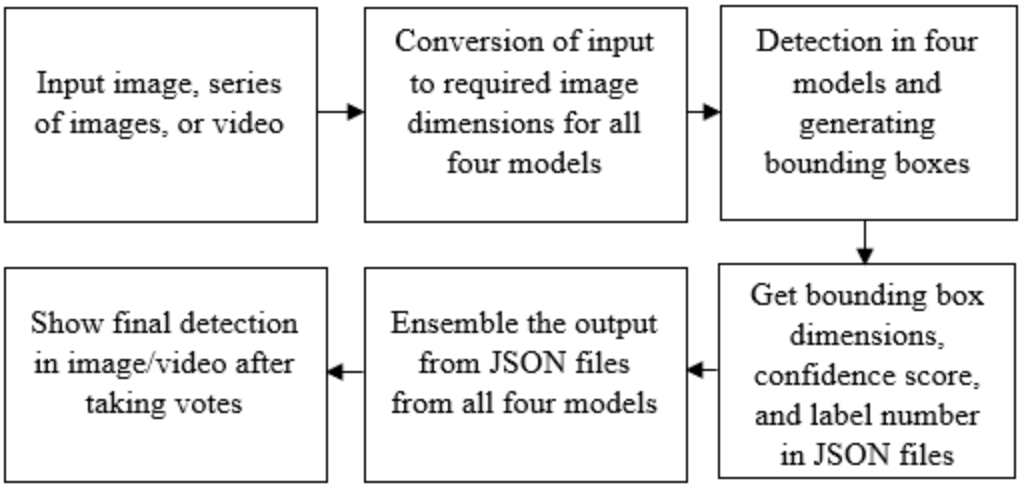


Figure 5.3.1 Series of steps in the Intelligent System

To perform all these steps together, we are using Google Colab Pro to run the 4 models one after the other and finally merging the output using the Weighted Boxes Fusion method to take the majority votes for prediction. Even if the bounding boxes created in each model are different, this method handles this variability and uses intersection over union to consider if the bounding box present is for the same object in all the models. Hence, if one model detects 4 cars, 1 pedestrian and other detects 3 cars and 2 pedestrians, then the IoU of all the 4 cars along with the other 3 cars will be calculated to decide the same detections. An IoU threshold needs to be predefined and set in the system and any value below that threshold will not be considered and take a negative vote in the ensemble output.

5.4 System Support Environment

5.4.1 Google Colab

Google Colab is a perfect platform for us to perform model training, testing, and ensembling due to its ease of sharing, powerful computing power(GPUs and TPUs) and fast training and testing time. To properly and efficiently train and test each of our models, we uploaded our dataset on Google Drive, performed model coding, training and testing on Google Colab. We continuously did hyperparameter tuning during the model training phase to achieve better performance. After adjusting models to their best performance, we ensembled our models on Google Colab. Besides, in this project, data augmentation, cleansing and transformation are mainly performed on Google Colab.

5.4.2 Jupyter Notebook

As a complement to the Google Collab, we also use Jupyter Notebook to train and test one of our models, SSD, because the Jupyter Notebook has better training stability than Google Colab. Besides, we statistically reviewed and summarized the model testing results on Jupyter Notebook with model performance graphs and tables for future demonstration and visualization.

5.4.3 Github

In this project, we set up a cohesive Github repository for our system to house all project's training data, testing data, module scripts and system output. Github repository has our log summaries of what was changed so we can better track our progress. Besides, different versions of updates were properly stored, so we can conveniently revert to an earlier version.

6. System Evaluation and Visualization

6.1 Analysis of Model Execution and Evaluation Results

For deep learning object detection model evaluations, mean average precision (mAP) is a common metrics for performance comparison across different models. In recent years, publications of object detection models and literature surveys often use the average precision (AP) score table format originated from the Microsoft COCO dataset evaluation methods. This section is a brief review of the common evaluation metrics including AP, mAP, intersection over union (IoU), confusion matrix, and area under curve (AUC).

In object detection models, for each predicted object there is an associated confidence score, or how likely the predicted class is correct. This confidence value is a probability value ranging from 0 to 1. However, a prediction is either correct or incorrect. Hence arises the need to convert the confidence probability into either positive or negative, and how. By setting a confidence score threshold, all predictions above the threshold are considered positive, and the rest negative. This way, a positively predicted class can be compared to the ground truth class.

In a confusion matrix, predictions are compared to ground truth, and the resulting categories are true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN). Precision score is defined as TP divided by TP plus FP, where TP + FP refers to the total positive predictions. Recall score is defined as TP divided by TP plus FN, where TP + FN refers to the total ground truth positives. With precision and recall scores calculated for a certain prediction set, we can plot a precision-recall curve. As the recall increases, precision can jump up and down. In order to minimize this effect, it is a common practice to interpolate the precision-recall curve. Average precision AP is defined as the area under the interpolated precision-recall curve. For each object class there is a distinct AP score; for example the original

COCO dataset has 80 classes and thus 80 AP scores. Mean average precision mAP is defined as the sum of all AP scores divided by the number of classes. In COCO style evaluations, COCO makes no distinction between AP and mAP, as it is understood that each class has its own AP, and the overall AP is actually mAP.

Intersection over union (IoU) is a score to evaluate how accurate the predicted bounding box location is compared to the ground truth bounding box location. IoU is defined as the intersection of predicted and ground truth bounding boxes divided by the union of predicted and ground truth bounding boxes. The higher the IoU score, the more accurate the bounding box location is. Similar to confidence scores needing a threshold to determine positive versus negative prediction, IoU also needs a threshold to determine whether a possible detected object should be considered as a positive detection. The IoU threshold is fundamental to the AP and mAP calculation just like the confidence threshold. For many object detection publications as well as this paper, the IoU threshold is set to be 0.5 unless otherwise specified.

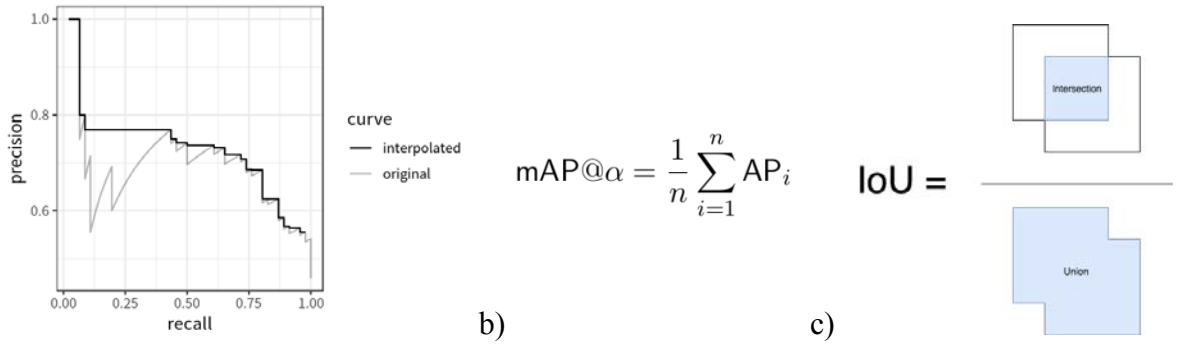


Figure 6.1.1 a) An example of an interpolated precision-recall curve. b) Formula for mAP, where α is the IoU threshold and n is total number of object classes. c) Illustration of IoU formula.

markings only will not detect cars, pedestrians etc. That is why we had to add additional object classes and re-do the annotations for the original images to include the additional classes.

Calculation of IoU is important in mAP metrics. For the our models built using the TensorFlow platform, including SSD, CenterNet, and EfficientDet, the default number of detections or bounding boxes is 100 per image. If we leave the 100 predictions as is, each image will have many redundant bounding boxes, or simply wrong predictions. For this reason, we apply non-maximum suppression (NMS) with IoU greater than or equal to 0.5, and a confidence filter to only include bounding boxes with confidence score higher than 0.3.

The use of different platforms in our custom trained models is a disadvantage when ensembling and calculating mAP metrics. Because each model generates predictions in different formats, we have written custom scripts in Python to standardize different formats. Thanks to the open source program Review Object Detection Metrics, we are able to save time from manually calculating mAP metrics using model predictions converted by our script. All of our custom trained models show predictions with IoU higher than 0.5, and varying confidence scores higher

than 26%. For Stage 1 Ensemble, we assigned different weights to our custom trained models, as some models can achieve better accuracy in general. Figure 6.1.3 below shows the final mAP for four of our models and Stage 1 Ensemble model. The mAP from models based on TensorFlow platform are inline with the official model mAP provided by TensorFlow. The Stage 1 ensemble mAP of 0.387 is higher than any of the four models that are used in the ensemble; the average recall (AR) scores in general are higher than the four models as well.

For mAP of Stage 2 Ensemble output, calculation is not possible with a test dataset that can showcase the ability of the model. To be more specific, our original test dataset is not annotated for the 80 COCO classes. The main point of our Stage 2 Ensemble model is to expand functionality to detect objects other than the 23 road markings classes and 9 helper classes. Therefore, we have decided to only include visualization for Stage 2 Ensemble and not mAP metrics.

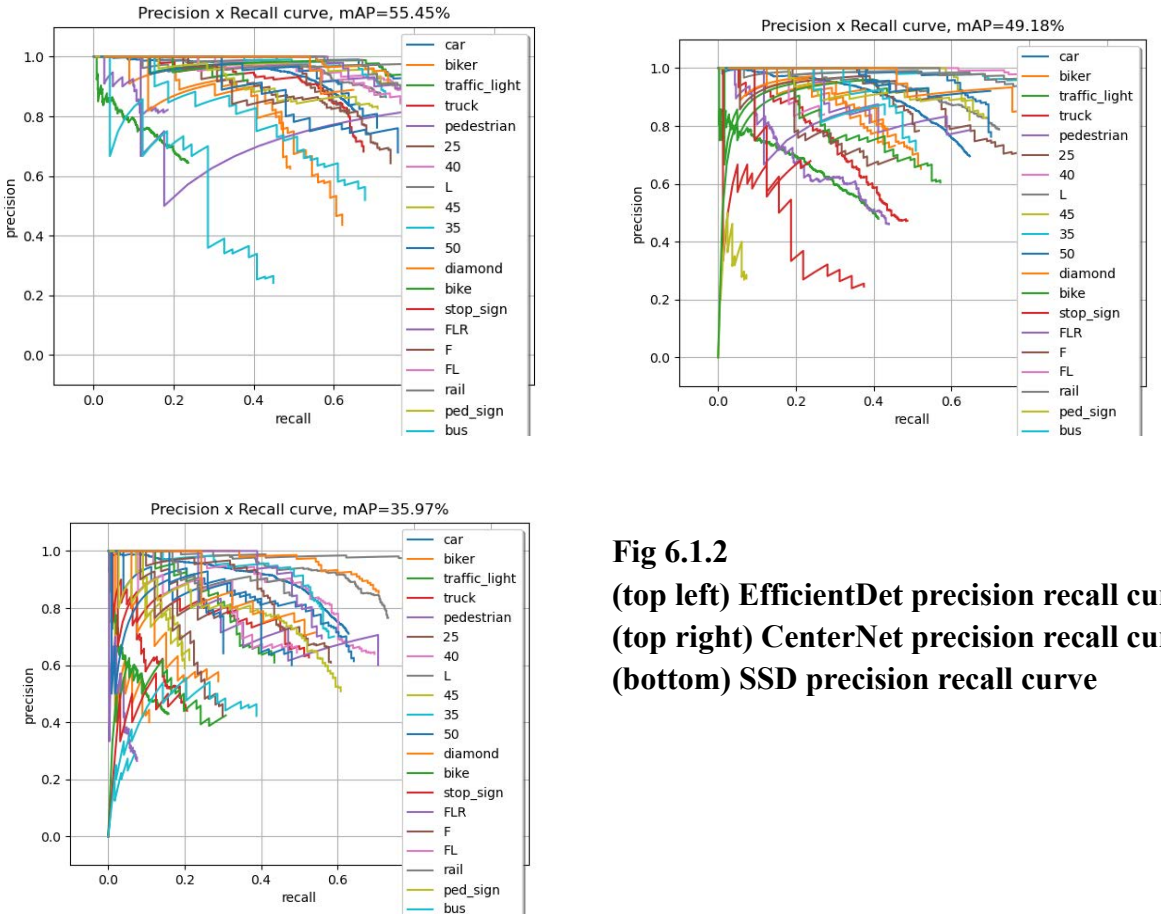


Fig 6.1.2
(top left) EfficientDet precision recall curve
(top right) CenterNet precision recall curve
(bottom) SSD precision recall curve

Custom YOLOv4 mAP

COCO METRICS:
AP: 0.3353602733558899
AP50: 0.7718821260485447
AP75: 0.2047219077564053
APsmall: 0.20951328734660393
APmedium: 0.37857282696119443
APlarge: 0.3615797083765879
AR1: 0.29718368339551515
AR10: 0.4232241705606323
AR100: 0.4234588677605454
ARsmall: 0.2618257603551286
ARmedium: 0.4623624776602961
ARlarge: 0.42860902090367814

Custom CenterNet mAP

COCO METRICS:
AP: 0.2519696410038126
AP50: 0.4892674264176864
AP75: 0.2187763108151276
APsmall: 0.1076868510610345
APmedium: 0.27745227693021857
APlarge: 0.2904955210658311
AR1: 0.22581246132057609
AR10: 0.30949606494951215
AR100: 0.30951573610210603
ARsmall: 0.13092848112049343
ARmedium: 0.340810666579900715
ARlarge: 0.3343539411783936

Custom EfficientDet mAP

COCO METRICS:
AP: 0.29075639356327576
AP50: 0.5515175247350308
AP75: 0.27810959945450975
APsmall: 0.05154362346968015
APmedium: 0.3088600812326196
APlarge: 0.4355062726274304
AR1: 0.2607688868723487
AR10: 0.3493969515586278
AR100: 0.34942565781472446
ARsmall: 0.054427178063593554
ARmedium: 0.36920760413090536
ARlarge: 0.5071614239330702

Custom SSD mAP

COCO METRICS:
AP: 0.18766240091320305
AP50: 0.3597897706020212
AP75: 0.17544543904168636
APsmall: 0.04256335064544277
APmedium: 0.2236228147492894
APlarge: 0.2030619162303574
AR1: 0.190409591639005
AR10: 0.2442793774147196
AR100: 0.2442854809303446
ARsmall: 0.05136340897428591
ARmedium: 0.2793278927918597
ARlarge: 0.266842403411449

Ensemble mAP

(Confidence threshold: 0.25
Weights:
YOLOv4: 2, CenterNet: 1.5,
EfficientDet: 1.5, SSD: 1)

COCO METRICS:
AP: 0.3872353337220656
AP50: 0.7279107534622377
AP75: 0.36487266312737554
APsmall: 0.17347364458807407
APmedium: 0.41712147490052354
APlarge: 0.4629853759649412
AR1: 0.323351944045684
AR10: 0.4458371575797668
AR100: 0.44591120468926215
ARsmall: 0.19912293679418158
ARmedium: 0.4815303470881397
ARlarge: 0.5122397981365311

Fig 6.1.3 mAP metrics for our custom trained object detection models.

6.2 Achievements and Constraints

The initial data processing steps involved multiple stages which were a huge achievement for our project as this dataset for roadmarks never existed before and gives a new dimension to the autonomous vehicle industry for their self-driving vehicle. The first achievement was collecting dataset using Google Restful APIs from maps, Street View, and cropping images manually from the internet for missing data classes. This task required a lot of effort and took a long time. The image 6.2.1 shows one of the manually collected images from Google Street View.

Once we completed collecting the dataset for all the roadmarks, the next step was to remove the imbalance in the dataset. To achieve this, we performed data augmentation techniques like rotation at different angles, zooming in randomly with different magnification, adding sun flares on the images to get day time scenarios, adding gravel patches on the roadmarks, as well as creating snow, and rain scenarios on the road. This made our dataset very unique resembling all the possible road scenarios and making our dataset that works for all the locations in the United States. The image 6.2.2 shows one such data augmentation achievement. After the data augmentation, we had a total of 6803 images of roadmarks which had more than 20,000 labels of different classes that were sufficient for training all roadmark classes incorporated in our project. This was another huge achievement for our targeted problems.

Fig. 6.2.1 Image collected using APIs

Fig. 6.2.2 Augmented Image to add rainy effects

After data augmentation, we manually labelled all the images in a YOLOv4 format which required a lot of effort and time. Manually labeling this many labels and images was very tedious and time consuming. This was another huge achievement for the project. By this stage, we had a complete dataset fully labelled for roadmarks for all the classes. Since we were also detecting cars, bikers, pedestrians, trucks, traffic lights, and traffic signs as our targeted problem, we needed data for the other remaining classes as well. For this, we collected different labelled datasets from the internet that constitute these classes. After working with such multiple datasets, we finalized the udacity self driving dataset to merge with the roadmark dataset.

The udacity self-driving dataset had 30,000 images out of which 400 from each class were selected manually. The dataset also had 10 original classes that were merged into 5 to meet our requirements. For example, green traffic lights and red traffic lights were merged into a single class traffic light. A script was developed that read each of the udacity label files and changed the label names from 0-9 which shows label number of its original classes to 23-28 which shows the label numbers for our classes. 0-22 are label numbers for roadmarks and 23-28 are labels for bikers, cars, pedestrians, trucks, and traffic lights. In this stage, a raw udacity dataset was manually selected to get required images, label files were changed using script to merge labels, another script to change label number to 23-28. After completion of this stage, we had two datasets for different classes for our targeted problem: (1) The complete roadmark dataset for 0-22 classes, (2) The edited udacity dataset for 23-28 classes. This required many python scripting and automation efforts and is one of the huge achievements in this project as it gives a complete dataset for our targeted problem.

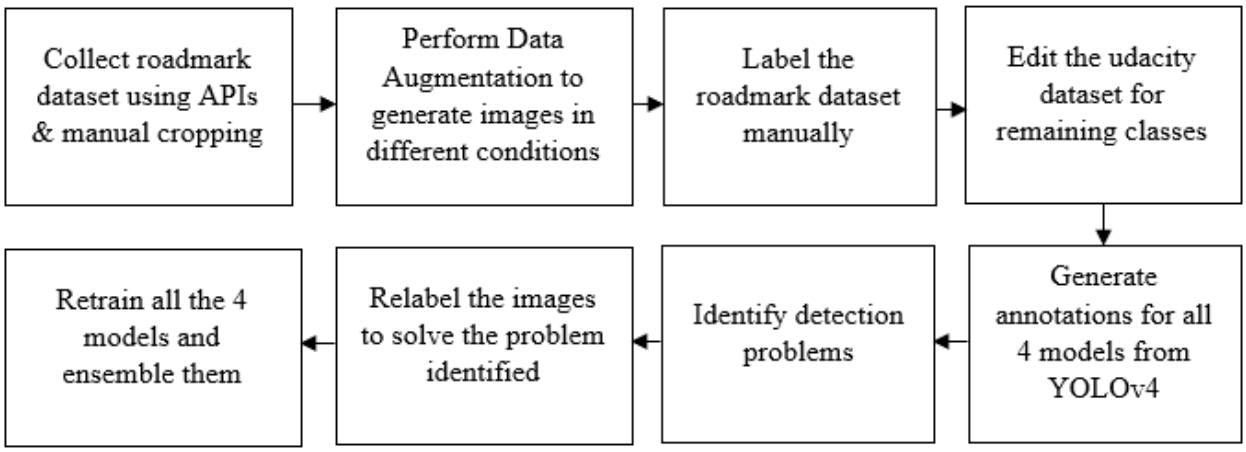


Fig 6.2.3 Achievements at various stages

Once, we had a complete dataset, we generated annotations for other models as well from the YOLO annotations. Annotations for SSD, Faster RCNN, and Mask RCNN were created to train different models. This required manual as well as scripting efforts and constituted one more achievement towards our project. Once we had label files in all the 4 formats, we trained our models and generated various outputs and exposed a new loophole in the output of all our models. Here, we found out that after the first few iterations in training the model, it would detect the cars and traffic lights as it had more frequency in the images and eventually change the weights to detect only roadmarks and forget weights to detect cars and traffic lights. Hence, after the first 500 iterations, the model would detect cars only and after 1000 iterations, the model forgets cars but only detects roadmarks. Eventually, after multiple trial and error methods, we found out the root cause for this problem. As the roadmark dataset did not have car images labelled, it was resulting in False Positives in the model training detecting cars that were not labelled in the actual dataset. This was resulting in changing the weights for cars as we train for

more iterations and forgetting the weights to detect cars. This resulted in a massive problem for us as we could not detect all the classes from our dataset.

Fig. 6.2.4 Detects only roadmarks

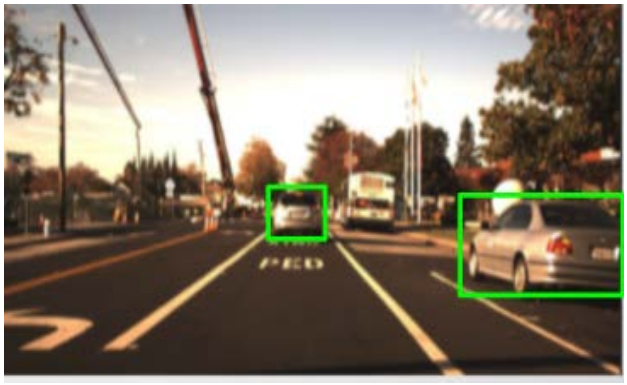


Fig. 6.2.5 Detects only cars

To overcome this problem, we relabelled all the roadmark images to include cars and relabelled all the udacity images to include roadmarks. This required a lot of time and manual effort. Once we relabelled the images, a script was created to merge the old labels with new labels which resulted in a single dataset rather than two. Finally, we resolved this issue and this was one of the greatest improvements as well as achievement in our project.

One other issue we faced was ensembling the multiple models we trained. Since all four models in our project have different system architecture, their prediction output is also very different. All the four models give bounding boxes in different formats (their label format). In some models we are getting .txt files and the other gives output in JSON files. Hence, we wrote scripts to extract bounding boxes, confidence of detection, and the label number from these files to perform ensembling. Once we get this information in the required format, the next step is to take votes for prediction. However, an object can be detected in multiple models to take votes but what if the bounding box created by these models are different. This was a huge problem that restricted the ensemble in our four models. To resolve this problem, we used a new method called Weighted Boxes Fusion method to take votes. This was the final achievement in this project after resolving the ensembling problem.

6.3 System Quality Evaluation of Model Functions and Performance

Three of the five major use cases for our object detection system on autonomous vehicles (AV) are for real world driving scenarios. That means streaming of video input and output, deep learning models inference, uploading and downloading from the cloud all required to be real-time. The status of running components is high in the priorities; we need to make sure every component is up and running. To monitor the status of all components, a heartbeat system is utilized. A heartbeat is a signal that each component will send to the heartbeat monitor every

minute. These components include the cameras, GPU if available on the AV, local storage drives, any connected cloud instances and services, and supplementary sensors for autonomous driving. Sending heartbeats regularly ensures the main computer is aware of the status of each component. If any heartbeat is missed, it will be logged. If four or more heartbeats are missed consecutively, the heartbeat monitor will report an error for the corresponding component. Then the main computer can try to restart the affected component and regain connection.

Under ideal circumstances, all components should run correctly when the AV starts and continue to run until the AV is shut down by a human operator. The heartbeat system ensures all components are online, however, it is possible for components to run into errors or to experience decrease in performance while being able to send heartbeats in a timely manner. To address such scenarios, we implement several performance metrics monitoring. The main responsibility of this object detection system is to run inference on input video and generate predictions in real-time. One major metric to monitor the correctness of this system is the real-time inference speed. Depending on the hardware responsible to run inference, the speed in frames per second (FPS) differs, making it less straightforward to set a universal FPS expectation for AVs equipped with different GPUs or cloud services with different accelerators. Despite the hardware differences, we can set the lower bound threshold FPS for each deep learning model. For example, YOLOv4 model should be able to achieve 30 FPS on higher end consumer grade GPU such as the Nvidia RTX 2070. Similarly, the SSD300 model should achieve 40 FPS. Each deep learning model reports the real-time inference speed in terms of FPS to the main computer. If one model performs slower than the corresponding threshold, an error is reported. Then the main computer can decide to wait and observe or restart a particular model and rely on other models for the meantime.

The integrated system connects to a cloud database and machine learning instances when the 5G signal is strong. While the 5G network is quickly covering the entire U.S., it is not uncommon for some areas to have poor connection due to various reasons. That means a stable 5G connection from the AV to cloud is not always guaranteed. To monitor the status of 5G connection, our system logs real-time 5G connect bandwidth, speed, and reception power. In the event of slow or unstable connection, data uploading or inference streaming is halted, and AV main computer will run inference on local hardware to ensure a continuous inference and output is not interrupted.

6.4 System Visualization

The AV system we designed was aimed to provide a comprehensive solution for autonomous driving but more specifically we are focusing on solving the road mark detection. The system can deploy a real-time bounding box with corresponding classification classes as outputs demonstrated by the open source library named opencv. We could either test the input testing image/video from the user or through the given live streaming video from the driver’s

dash camera. As the reported accuracy from the previous section in 6.1. We see that the four different models can generate a relatively accurate prediction for most of the labels. The model can successfully detect the sign such as STOP nearly 95% over time with a precise bounding box from different segmentation such a technique we used for augmentation purposes. However, the system has reached some degree of limitation on the image with no clear sign that has been shown inside the image. Such as the label output for Pet and Crossing is very low in accuracy.

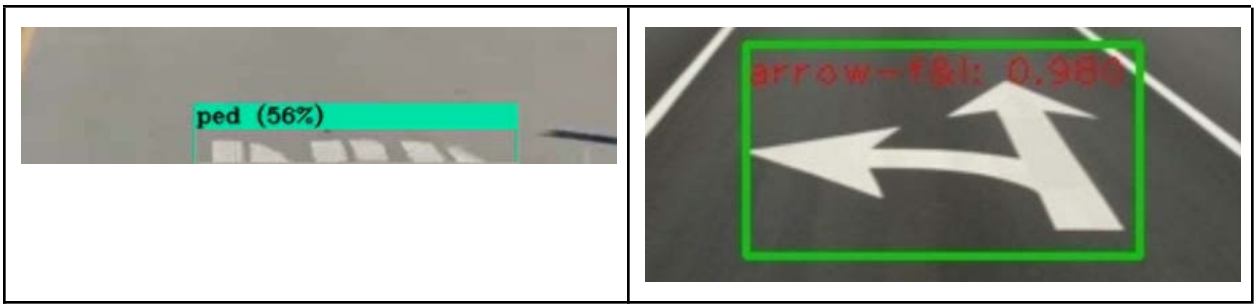


Fig 6.4.1 Detections by the system

In the last stage we hope we could build an application on a webpage which allows the pre-trained model to iteratively test with people's image or output by the API we provided. The main purpose of this deployment is for the improvement of user exploration with a foretaste of autonomous driving pilot systems. The testing portal should be simple and friendly to play around for people with non technical background to understand how the system is functioning. And the users page would have the link to people with experts who can gain full access to our open source codes and data from end to end replication and research support.

7. Conclusion

7.1 Summary

A fully self-driving car would be able to detect and classify all the mobile objects on the road along with traffic regulatory objects. To achieve the same, past researchers have developed many datasets so that it can be used for object detection. However, none of these datasets had all the traffic regulatory road markings that are found in the US. Hence, the ultimate goal of this research was to develop a system that detects roadmarks along with cars, bikers, pedestrians, traffic lights, and traffic signs as a one step ahead of the past research work.

To fulfill this goal, we started with collecting images for various roadmarks that can be used for training our models for object detection. This was a tedious task as we had to collect thousands of images manually from internet sources and using the dash cameras on our cars. After collecting a sufficient number of images, we decided on 23 roadmark classes that cover almost all road markings found in the US. After collecting images, we performed data

augmentation to increase the number of images by changing the orientation of the images, randomly zooming in and out, changing the brightness, and introducing various weather conditions to make the model robust so that it can detect objects in different weathers as well as any time of the day. After completing the augmentation, we had a sufficient number of images for each of the roadmark classes. Hence, all these images were manually annotated to get a complete fully annotated roadmarking dataset. By this stage, we had a complete dataset for main classes but we still needed images for pretrained classes. Hence, we preprocessed the Udacity Self-Driving Car dataset to get the images and annotations in YOLO txt files for our model. This resulted in images and annotations for 32 classes. As a result of collecting and annotating these images, a complete dataset was formed for all the mobile objects on the road including roadmarks which was one of the aims of this project.

Once we had the complete dataset, we started training our five different models namely YOLOv4, SSD, EfficientDet1, Faster RCNN, and CenterNet. On training we found that the model would not retain weights for all the objects but keeps on forgetting it as the pretrained objects were not labelled in the roadmarking dataset and the roadmarks were not labelled in the Udacity images for pretrained classes. Hence, to overcome this problem, we relabelled the images for the missing classes in both the dataset and reformed the whole dataset by writing multiple scripts and merging the new and existing labels. Once relabelling was done, we retrained all the five models which resulted in successful detection of all the classes at the same time. This was another achievement with individual models able to detect roadmarks which was never done before along with other road objects.

Since we had all the five models, the next step was to perform stage 1 ensembling them and improve the detection accuracy and mAP. To enable this, we chose the five best models in terms of mAP and detection time, and ensembled them by using Weighted Boxes Fusion method. To ensemble, our system takes output in different models, then generates output files which are then voted according to the weightage provided to generate an improved ensembled model. After completing the stage 1 ensembling, our model was able to detect objects with improved mAP and better accuracy.

The last step was to perform stage 2 ensembling to merge the outputs from a stage 1 ensembled model and the pretrained YOLOv4 on COCO dataset. This resulted in additional 90 classes apart from the 32 classes from stage 1 ensembled model. Apart from stage 2 ensembling, the model was also able to detect cats, dogs, and many other small objects that can be found sometimes on the road. This was another achievement for this research project. Hence, after completing all these steps, a two stage ensembled model with high mAP was achieved that detects almost any object found on the road.

7.2 Benefits and Shortcoming

There are numerous benefits of the two staged ensembled model that we have developed. As it is able to detect and classify 122 classes, it can be used as an eye to an autonomous vehicle which will detect all the objects on the road and the system can interpret the signal to take the necessary traffic regulatory steps for the vehicle. This system does all the detection for autonomous vehicles and only leaves the mechanical part to be merged with the system along with few modifications. The system that we have developed is also more robust than the past research work as it is ensembled with different types of models due to which it takes all the advantages and overcomes the disadvantages of these deep learning algorithms. Another benefit of the system is the second stage stacking to merge pretrained COCO classes which adds 90 new classes such as cats, dogs, trains, buses, and airplanes. The model also uses four state of the art deep learning algorithms which leads to very high mAP and high efficiency which is a benefit in comparison to the past researches in this domain.

Even though it has many benefits, there are some shortcomings to the model. As the model does all the tasks for object detection, it does not do all the tasks for an autonomous vehicle. For example, an autonomous vehicle system will also detect the lanes and give inputs for lane changing. This can be done by segmenting different lanes and identifying the pavements so that all the lanes on the road are considered. Another shortcoming of the system would be that it does not calculate the distances of the objects detected. If the distances are calculated then the system can interpret the exact location of the objects detected and can work smarter than the current system. These distances can be calculated by counting the pixel values for the standard traffic regulatory signs and using lidar for vehicles. If these two shortcomings are added to the existing ensembled model we developed, then it can help in developing an autonomous vehicle.

7.3 Potential System and Model Applications

The application spanned to a large domain of real world applications such as public transportation, sanitation operations, ports and terminals, mining, retails etc.

One scenario could be: AI-driven autonomous vehicles as designated for Covid-19 assessment in hospital or quarantine process can be a potential application to solve the problem of direct contact with tested positive patient in case of food and supply or any direct contact with contaminated personal items so that the contaminated transports can be thoroughly sanitized and the health practitioners also can have enough reaction time to be awarded.

Haidilao, the first AI hot pot restaurant in China is one well known successfully deployed business case with AI robots for food delivery in the catering industry. The real-time object tracking which mainly to reduce labor cost has been successfully solved and shown during peak hours with human runners around. The automated food delivery system is not only accurate to present food to each destination but also combined with matured situational awareness and response systems in case of traffic or blocked by customers ahead of the route.

7.4 Experience and Lessons Learned

The lessons that we learned from this master project are mainly in four different aspects.

1. Post- reviews: We learned that we need to have a thorough and mindful review after each milestone has been completed. The feedback from Dr. Gao is critical for us to modify our past draft. Closely follow with the rubric for which criteria we left behind that is where we need to improve on.
2. Team meetings: We learned from the experience of lack of communication at the very beginning of this project and realized the importance of bringing each of us to the same level of understanding. In the meanwhile, we need to help each other to unblock any obstacles that slow them down. And listen to each other's thoughts. Regular meeting on a weekly basis is the key for us to reach our goal.
3. One-to-one meetings: The one-to-one meetings with our team members or Dr.Gao is the most effective way to succeed. We take this opportunity to ask direct questions and helpful supervision in a timely manner. This experience helps our team not only to accelerate our model deployment but even more in very detailed programming blocks.
4. Self-learning and Research skills: The training of the ability to search for what we need is shown on each one of us from diving into our designed models. The improvement from knowing nothing to a detailed presentation is remarkable throughout this project. As an undergraduate student, having the skills of research and learning are critical skills that we have been practicing which could benefit us in our future careers.

7.5 Recommendations for Future Work

In this project, we used several deep learning algorithms like YOLO, SSD, Faster R-CNN and ensemble models to detect and recognize road markings, traffic signs, cars, pedestrians and traffic lights in real-time. From the viewpoint of object detection accuracy(mAP) and algorithm time-consuming, our proposed deep learning algorithm and ensemble models have remarkable advantages. Firstly, our deep learning algorithms and ensemble models considerably enhance the driving safety of intelligent vehicles in the actual driving environment, and secondly, they effectively meet the real-time requirements of self-driving cars and intelligent vehicles. Furthermore, a powerful technical guarantee is provided for the steady development of autonomous driving.

Despite excellent performance of our proposed approach there is still room for improvement. Our analysis revealed that the detection and recognition accuracy of mobile objects on the road can be further improved, and the classes of traffic regulatory objects to be detected can be further increased. Future work should focus on improving these parts of the system.

7.6 Contributions and Impacts on Society

From the educational perspective, our project can be helpful for people who are learning how to drive cars. With real-time detection and recognition of traffic signs, road markings, bikers, pedestrians and cars, new drivers would notice and memorize the traffic regulatory objects faster in the real driving environment. Without rote learning the regulatory objects from the driver's manual again and again, new drivers now have a more efficient and practical way to learn driving. Furthermore, the existing deep learning model and datasets combine the European and worldwide traffic environment. Our project specializes in the U.S traffic environment, and thereby can be a good tool for new drivers in the U.S.

From the economic perspective, self-driving technology is extremely costly to develop. In 2020, Waymo announced that they have spent an estimated $3.5 billion on financing its self-driving R&D. In our project, we used relatively less economical resources to solve a small problem in the self-driving area. We used UCloud and Colab Pro for model construction, model training and testing, locally we used Tensorflow 2.0 and Pytorch 1.5. In data collection, we also chose less costly solutions such as Google Earth, Google Street View, web crawlers, scraping technologies and dashcam. We hope to develop an economical friendly resolution to assist the self-driving technology, such as the lane-keep assist.

From the social wellbeing perspective, although autonomous vehicles potentially improve energy consumption of traditional gas based vehicles and therefore reduce carbon emissions, rebound effects could mitigate this effect due to uptake in use of road vehicles. Self-driving vehicles will also play a central role in the future to advance ride-sharing concepts.one benefit is that people do not need to worry about parking the fully autonomous vehicle, as optimal capacity utilization with as little standstill as possible means the driverless car can do it for itself. A more efficient deposition significantly reduces the need for parking spaces in urban areas that can then be used in a variety of other ways. Car sharing, flexible commuting times and working from home are therefore key considerations to optimize the impact autonomous vehicles may have.

References

1. Wang, G.Y.; Ren, G.H.; Jiang, L.H.; Quan, T.F. Hole-based traffic sign detection method for traffic signs with red rim. Vis. Comput. 2014, 30, 539–551.

2. Hechri, A.; Hmida, R.; Mtibaa, A. Robust road lanes and traffic signs recognition for driver assistance systems. Int. J. Comput. Sci. Eng. 2015, 10, 202–209.

3. Xiao, Z.T.; Yang, Z.J.; Geng, L.; Zhang, F. Traffic sign detection based on histograms of oriented gradients and boolean convolutional neural networks. In Proceedings of the 2017 International Conference on Machine Vision and Information Technology (CMVIT), Singapore, 17–19 February 2017; pp. 111–115.

4. Sun, Z.L.; Wang, H.; Lau, W.S.; Seet, G.; Wang, D.W. Application of BW-ELM model on traffic sign recognition. *Neurocomputing* **2014**, *128*, 153–159.

5. Qian, R.Q.; Zhang, B.L.; Yue, Y.; Wang, Z.; Coenen, F. Robust Chinese traffic sign detection and recognition with deep convolutional neural network. In Proceedings of the 2015 11th International Conference on Natural Computation (ICNC), Zhangjiajie, China, 15–17 August 2015; pp. 791–796.

6. He, K.M.; Zhang, X.Y.; Ren, S.Q.; Sun, J. Deep residual learning for image recognition. In Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.

7. Yuan, Y.; Xiong, Z.T.; Wang, Q. VSSA-NET: vertical spatial sequence attention network for traffic sign detection. *IEEE Trans Image Process* **2019**, *28*, 3423–3434

8. Xiao, Liang & Chuanxiang, Li & Zhao, Dawei & Chen, Tongtong & Dai, Bin. Road Marking Detection Based on Structured Learning. **2016**. 10.1109/WCICA.2016.7578824.

9. S. P. Rajendran, L. Shine, R. Pradeep, and S. Vijayaraghavan, "Real-Time Traffic Sign Recognition using YOLOv3 based Detector," 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 2019, pp. 1-7, DOI:10.1109/ICCCNT45670.2019.8944890

10. J. Zhang, S. Hu, and H. Shi, “Deep Learning based Object Distance Measurement Method for Binocular Stereo Vision Blind Area,” *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 9, 2018.

11. A. Campbell, A. Both, and Q. (C. Sun, “Detecting and mapping traffic signs from Google Street View images using deep learning and GIS,” Computers, Environment and Urban Systems, vol. 77, p. 101350, 2019.

12. Muthalagu, R., Bolimera, A., & Kalaichelvi, V. (2020). Lane detection technique based on perspective transformation and histogram analysis for self-driving cars. *Computers & Electrical Engineering, 85*, 106653. doi:10.1016/j.compeleceng.2020.106653

13. S, S. J., & P, E. R. (2021). LittleYOLO-SPP: A delicate real-time vehicle detection algorithm. *Optik*, *225*, 165818. https://doi.org/10.1016/j.ijleo.2020.165818

14. Yang, W., Li, Z., Wang, C., & Li, J. (2020). A multi-task Faster R-CNN method for 3D vehicle detection based on a single image. *Applied Soft Computing*, *95*, 106533. https://doi.org/10.1016/j.asoc.2020.106533

15. Lin, Tsung-Yi, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2014. "Microsoft COCO: Common Objects in Context," May. http://arxiv.org/abs/1405.0312.

16. LISA: Laboratory for Intelligent & Safe Automobiles Traffic Sign Dataset, Professor Mohan M. Trivedi. LISA-CVRR-UCSD, http://cvrr.ucsd.edu/LISA/contact.html

17. Introducing the Mapillary Traffic Sign Dataset for Teaching Machines to Understand Traffic Signs Globally, Jun 27 2019, Christian Ertler, https://blog.mapillary.com/update/2019/06/27/mapillary-traffic-sign-dataset.html

18. "A Practical System for Road Marking Detection and Recognition", Tao Wu and Ananth Ranganathan, IEEE Intelligent Vehicles Symposium, 2012. http://www.ananth.in/RoadMarkingDetection.html

19. Arcos-García, Á., Álvarez-García, J. A., & Soria-Morillo, L. M. (2018). Evaluation of deep neural networks for traffic sign detection systems. *Neurocomputing*, *316*, 332–344. https://doi.org/10.1016/j.neucom.2018.08.009

20. Choi, J., Chun, D., Kim, H., & Lee, H.-J. (2019). Gaussian YOLOv3: An Accurate and Fast Object Detector Using Localization Uncertainty for Autonomous Driving. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. https://doi.org/10.1109/iccv.2019.00059

21. Dai, J., Li, Y., He, K., & Sun, J. (2016). R-FCN: Object Detection via Region-based Fully Convolutional Networks. *ArXiv, abs/1605.06409.*

22. Jin, Y., Fu, Y., Wang, W., Guo, J., Ren, C., & Xiang, X. (2020). Multi-Feature Fusion and Enhancement Single Shot Detector for Traffic Sign Recognition. *IEEE Access*, *8*, 38931–38940. https://doi.org/10.1109/access.2020.2975828

23. Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934. https://arxiv.org/abs/2004.10934

24. Lin, Tsung-Yi, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2014. "Microsoft COCO: Common Objects in Context," May. http://arxiv.org/abs/1405.0312.

25. Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. ICML, 2019. https://arxiv.org/abs/1905.11946

26. Mingxing Tan, Ruoming Pan and Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. CVPR 2020. https://arxiv.org/abs/1911.09070

27. R. Solovyev, W. Wang, and T. Gabruseva, “Weighted Boxes Fusion: Ensembling boxes from different object detection models,” *Image and Vision Computing*, vol. 107, p. 104117, 2021.

28. R. Padilla, W. L. Passos, T. L. Dias, S. L. Netto, and E. A. da Silva, “A comparative analysis of Object Detection Metrics with a companion open-source toolkit,” *Electronics*, vol. 10, no. 3, p. 279, 2021.

Appendices

Appendix A – System Testing

The developed system has a two stage ensembled model which generates output at multiple stages during inference. First, the output from each separate model is generated which is passed into stage 1 ensemble model to merge the output for improving the detections. This ensembled model is again stacked with the COCO model to detect more objects in stage 2 ensembling making the total classes to 122. The test results from different stages to detect various objects are shown below:

Fig. A1 Detection of cars, road marks, bikers, and pedestrian corssing sign

Fig. A2 Detection of traffic lights, multiple pedestrians, and Left turn at intersection

Fig. A3 Detection of STOP sign and STOP road mark

Fig. A4 Detection of multiple Bikers

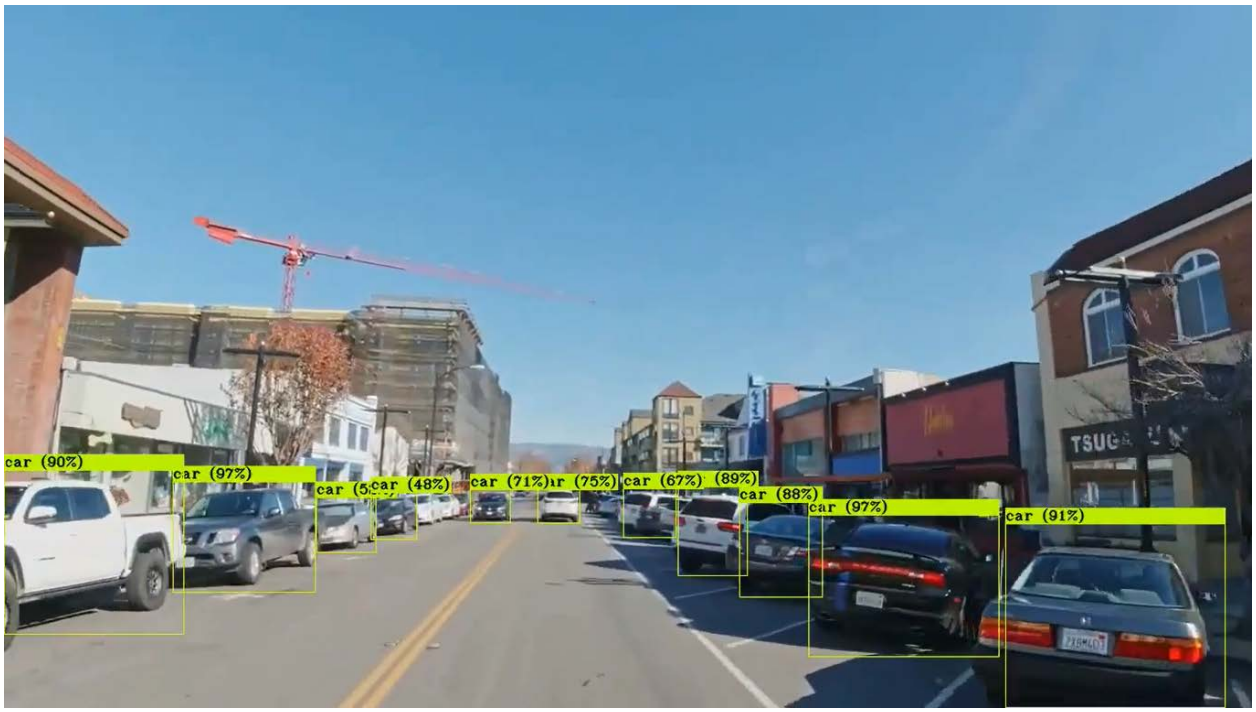


Fig. A5 Detection of multiple cars

The above shown images are only a few of the 122 classes that the model can detect. These detections were taken from a video passed in our ensembled model. These detections show how the model does not miss any trained objects and classifies them very accurately. As all the models generate text files for each frame, the ensembling is very robust and the settings for ensembling can be changed very easily to change the voting power and weightage of each model in the ensembling process. The 122 classes in the ensembled model covers almost all the objects that are found on the road that are required for a self-driving vehicle.

Appendix B – Project Data Source and Management Store

The training data for roadmarks was manually collected from videos using dash cams on our cars, and internet sources such as Google maps, and Google Earth as discussed in section 3.2 for data collection. The collected images were cropped according to the requirements and merged with an image dataset collected by Tao W. and Ananth R. (2012) which had few images for some of the classes. However, these images were not annotated in the required format due to which we merged these images with our manually collected data. These images were augmented with various transformations to meet the requirements for training and validation. On meeting a satisfactory number for each class, the dataset was manually annotated to get a fully labelled dataset for 23 roadmarks which never existed before. Image A1 shows the output of the manually annotated image in the YOLO txt format.

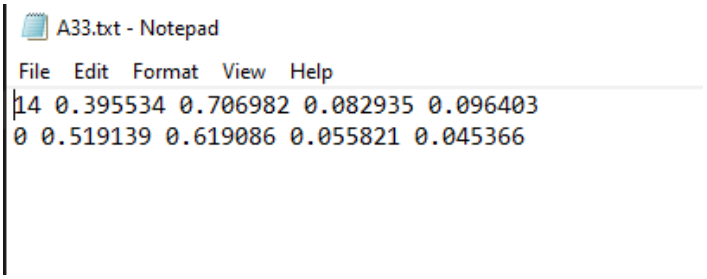


Fig. B1

The link to the pre collected roadmark images:
http://www.ananth.in/RoadMarkingDetection.html

For the pretrained classes, we used the Udacity Self Driving Car dataset. As we had annotated our roadmark images in YOLO txt format, we used Roboflow to get the Udacity annotations in the same format. The Udacity dataset can be found at:
https://public.roboflow.com/object-detection/self-driving-car

Once it was merged with the manually collected dataset, the annotations were changed as per the label number required for our models. The image B2 shows the conversion from original Udacity labels to the labels required for our model. Once we had all the images in YOLO txt format, we wrote scripts to convert it into VOC Pascal, and TensorFlow TF which are the required input formats for our additional models like SSD, EfficientDet1, Faster RCNN, andCenterNet. The image below shows the annotation converted from YOLO txt to Pascal VOC xml file which is the required input for two of our models.

```
0 0.4361979166666667 0.48083333333333333 0.022395833333333334 0.04
0 0.2734375 0.49083333333333334 0.028125 0.03166666666666667
3 0.28072916666666664 0.44083333333333335 0.011458333333333333 0.028333333333333332
1 0.3619791666666667 0.495 0.04479166666666667 0.056666666666666664
3 0.38385416666666666 0.3983333333333333 0.009375 0.03
1 0.39739583333333334 0.485 0.028125 0.03666666666666667
1 0.39739583333333334 0.4775 0.023958333333333335 0.025
3 0.4270833333333333 0.3908333333333333 0.010416666666666666 0.028333333333333332
3 0.42916666666666664 0.38916666666666666 0.0125 0.03166666666666667
3 0.5135416666666667 0.4116666666666667 0.016666666666666666 0.03333333333333333
1 0.5916666666666667 0.4708333333333333 0.06666666666666667 0.055
2 0.5817708333333333 0.495 0.017708333333333333 0.07666666666666666
```

Fig. B2 Annotation for pretrained classes in Udacity Dataset

```
23 0.4361979166666667 0.48083333333333333 0.022395833333333334 0.04
23 0.2734375 0.49083333333333334 0.028125 0.03166666666666667
26 0.28072916666666664 0.44083333333333335 0.011458333333333333 0.028333333333333332
24 0.3619791666666667 0.495 0.04479166666666667 0.056666666666666664
26 0.38385416666666666 0.3983333333333333 0.009375 0.03
24 0.39739583333333334 0.485 0.028125 0.03666666666666667
24 0.39739583333333334 0.4775 0.023958333333333335 0.025
26 0.4270833333333333 0.3908333333333333 0.010416666666666666 0.028333333333333332
26 0.42916666666666664 0.38916666666666666 0.0125 0.03166666666666667
26 0.5135416666666667 0.4116666666666667 0.016666666666666666 0.03333333333333333
24 0.5916666666666667 0.4708333333333333 0.06666666666666667 0.055
25 0.5817708333333333 0.495 0.017708333333333333 0.07666666666666666
```

Fig. B3 Annotations after converting Udacity Dataset labels for training

```xml
<annotation>
  <folder>VOC2007</folder>
  <filename>A33.jpg</filename>
  <source>
    <database>Coco database</database>
  </source>
  <size>
    <width>1280</width>
    <height>720</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>Arrow-Left</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>453</xmin>
      <ymin>474</ymin>
      <xmax>559</xmax>
      <ymax>543</ymax>
    </bndbox>
  </object>
  <object>
    <name>Speed-25</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>628</xmin>
      <ymin>429</ymin>
      <xmax>700</xmax>
      <ymax>462</ymax>
    </bndbox>
  </object>
</annotation>
```

Fig. B4 Training data in Pascal VOC format in xml files

We also collected videos using our dash camera to test our models on the streets of San Jose. After detection results from each model, the outputs were generated in txt files so that it can be tested for mAP, ROC curves, and then ensemble in our ensemble model. The image B5 shows the output results in the form of label number, Confidence score, and the next four values are the values to create bounding box on the image in form x, y, w, h where x and y are the pixel coordinates whereas w is width and h is height of the image size.

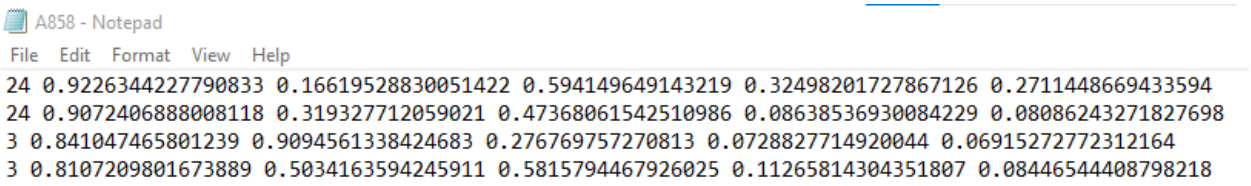


Fig. B5 Output file for test images

These files were generated for all the trained models to ensemble it and further used for evaluation to calculate mAP for each model. The training data, test data, output results in txt from all the models, and image annotation in all formats have been uploaded on the shared Google Drive folder.

Google Drive Link:
https://drive.google.com/drive/folders/1cA4uBgwc0sVnUubM5PhNPG_8ty0fh4Ix?usp=sharing

Appendix C – Project Program Source Library, Presentation, and Demonstration

The project involved development of multiple scripts for data collection, data preprocessing, data augmentation, training, ensembling, testing, and evaluation. Various augmentation tools like Automold Road Augmentation and Keras Preprocessing library were utilized to generate the required number of images. Multiple scripts were written to get the Udacity dataset in the required format for our models. Scripts to convert all the annotations in different formats for different models were created. For our two stage ensembling model, we developed additional programs to stack an already ensembled model with other COCO classes which created 122 classes (32 from our dataset + 90 from COCO dataset) classes after stage 2 ensembling. To run videos on the ensembled model, we developed programs that convert videos into frames for running the ensembled model reading the output txt file for each frame and then generating bounding boxes on each image. Once all the images with bounding boxes are generated, the images are again merged to generate a video enabling our two stage stacked model to run on videos as well.

The tool we used to calculate the evaluation metrics can be found at:
https://github.com/Cartucho/mAP

The coding library used for data augmentation was taken from GitHub:
https://github.com/UjjwalSaxena/Automold--Road-Augmentation-Library

The various ensembling methods that we have used for our two stage ensembling model was found on GitHub. We developed multiple scripts to get the required output from our model in a way that it is accepted by this library called ‘ensemble-boxes’. We tried multiple ensembling methods found in this library:
https://github.com/ZFTurbo/Weighted-Boxes-Fusion

All the program codes used and developed for training, data engineering, ensembling and testing as explained above can be found on the Google Drive folder along with the presentations, demonstration videos, and reports associated with the project.

Google Drive link:
https://drive.google.com/drive/folders/1cA4uBgwc0sVnUubM5PhNPG_8ty0fh4Ix?usp=sharing