Smart Traffic Light with Emergency Vehicle Prioritization

Azevedo, Jonathan (BS Electrical Engineering)
Fricker, Sean (BS Electrical Engineering)
Hou, Poliva (BS Electrical Engineering)
Pham, Elizabeth (BS Electrical Engineering)

Introduction

In this project, we propose a Smart Traffic Light system with Emergency Vehicle Prioritization (STLEVP). The STLEVP is a portable traffic light system that can detect and prioritize emergency vehicles on main roads. We acknowledged that emergency vehicles are not the most common sight so we customized our project to also gather data, in order to provide normal traffic efficiency during non-emergency situations. Using deep learning, the system can recognize emergency and normal road vehicles in a real-time environment. The deep learning that we used, implemented Pre-trained and Self-trained models, as well as Sort and Virtual Lines. The Raspberry Pi 4 is used as the hardware platform because of its compact and powerful computational ability. It is capable of running the object detection software that we customized and implemented into Tiny YOLO. However, the size and computational ability of the Raspberry Pi does result in a very low frame rate per second. We ran this model with pre-recorded videos due to the covid situation.

Methodology

The first step to designing the smart traffic light was to identify and research the issues of the existing traffic light systems. We took our inspiration from pre-existing image detection systems in order to create our hierarchy.

1. **YOLO**
2. **Sort**
3. **Virtual Line**
4. **Emergency and Normal Road Vehicle Detection**

With TensorFlow acting as our Top function, we customized and added to the YOLO v3 Deep Learning to create our image detection system. Once we were able to identify objects, such as cars and trucks, accurately, we moved onto to data gathering and analysis of the traffic. We edited and developed our own iteration of the popular counting module Sort. Through Sort we were able to use the concept of Virtual Lines and set two lines, one

at the traffic light intersection and the other two car lengths back from the intersection line. With these two virtual lines we were able to collect data on car density and the wait time of vehicles. Our pre-recorded video contained five lanes, and so the traffic density evaluated if there were 10 cars present between the two virtual lines in order to determine if traffic was heavy. For the wait time, we timed each individual car (which is identified by number with Sort) and recorded the time it took for each car to cross both lines. If this time eclipsed our minimum limit (20 seconds) it was added to a .csv that is created every hour. Based on this data we were able to create average time graphs.

Importing necessary packages and libraries:
- numpy
- scipy
- pillow
- scikit-image
- pandas
- seaborn
- utlfit3

OpenCV and TensorFlow was installed onto Raspberry Pi 4. OpenCV is a library dedicated at real-time computer vision and TensorFlow is a machine learning platform that focuses on deep neural networks and training. In order to ensure the code runs properly, the tiny weights from pjreddie were downloaded. The camera was configured on Raspberry Pi 4. Lastly, a sample video was ran through to test the code. Once the code was running properly, we created our own custom database images for necessary classes that are essential to our needs. We decided to create four classes - ambulances, buses, cars, and trucks. There are three datasets - training, validation, test. The images were downloaded from OpenImageV6.com. For each image, we manually created a box over the desired pixels (of the corner of the boxes) in Ms Paint. Afterwards, a folder was created to store the four classes. For each class folder, the image was saved as the name of the image, the four coordinates of the labeled boxes, and the class code, all separated by commas. After collecting and labeling the images, the model was trained to the custom dataset. The batch size was set to 4 and the number of epochs was set to 50. This training took approximately 15 hours.

Once the training is completed, the mean average precision was calculated and the total loss was plotted. A sample clip is ran through to see the results.

Analysis and Results

After importing and installing the necessary libraries, packages, YOLOv3, and the pretrained weights onto the Raspberry Pi 4, the unprocessed video was ran through the code. The results are shown in the QR code below.

As the graph shows, the loss is initially decreasing; however, the loss begins to increase after ten epochs. This indicates that the model is starting to overfit the data, meaning that the model can only detect ambulances to our custom data. In the first few seconds, the model detected the ambulance as both truck and ambulance. Non-max suppression can help reduce the error. Therefore, for the second training, we decided to apply early stopping to our model to stop the overfitting. We changed the epochs from 50 to 10. The results from applying the early stopping method can be seen below.

By applying the early stopping method, the loss improved from 7.93 to 5.23 and the mean average precision improved from 65.06% to 75.35%. When analyzing the Wait Time data we took at the intersection of Tully and Capitol Expressway, we initially assumed that the Expressway would have a lesser wait times at red lights. However, as we saw, the priority given to the Expressway was given in terms of how long the light stayed green instead of how long the light stayed red. We intended to use this data to show how we could improve and help speed up the changing of the lights during low traffic hours. However, we couldn’t properly teach the system to check if the opposing traffic was empty before collecting data.

Summary/Conclusions

The goal of this project was to create a system that could use deep learning architecture to detect and identify the ambulances and give them safe priority in traffic. By using the early stopping method, while identifying our emergency and normal vehicles, the average precision of vehicles detection improved from 65.06% to 75.35%. We found that many of our obstacles arise from the camera angle that we chose (on top a building). We believe that if we had the optimum placement of the system (on top the traffic light) and better hardware options (multiple Raspberry Pis and the Movidius NCS usb option) we could more accurately collect data and help emergency and normal vehicles achieve greater safety.

Acknowledgements

We would like to thank professor Wong for his input and patience. We would also like to thank Rokas Balys for detailed explanation on debugging.

Key References


