Object-Detection-in-an-Urban-Environment
Zifeng Wu

Github link: https://github.com/zwu620/Object-Detection-in-an-Urban-Environment.git

Project Overview
This project aims to have students to train a convolutional neural network (CNN) based on a pre-trained network to detect and classify objects using data from the Waymo Open Dataset. The CNN model is used to identify cars, pedestrians, and cyclists from the provided dataset containing images of urban environments with annotated cyclists, pedestrians, and vehicles. The object detection is an important component of self driving car systems because this is the "eye" of a self-driving car so it can “see” the surrounding and therefore take the correct action. For example, slow down if there is a preceding vehicle or stop when there is a pedestrian crossing the road.

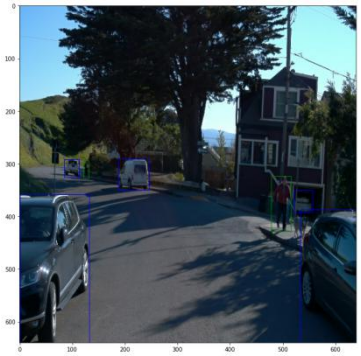
Set up
Had some issues to set up on my local computer. This project was done in the workspace provided by Udacity. Due to the space constraints and resource exhausted issues. The results are limited in terms of 1) the number of steps for the training and 2) the number of checkpoints can be included in the evaluation.

Dataset

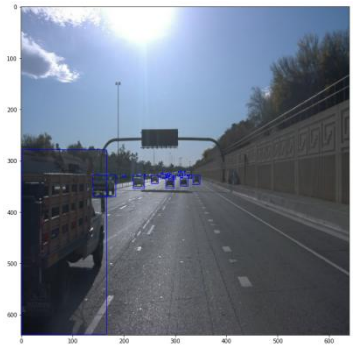
Dataset Analysis
The provided Waymo dataset includes a variety of environment. Examples include 1) very dense downtown including heavy vehicle and pedestrian volumes, 2) low-volume minor arterial or local collectors, 3) high-volume freeway. Below are example images corresponding to the three environments. A total of 10 images were saved to the folder: **10_example_images**. The weather conditions range from strong sunlight to cloudy or rainy conditions. Images were taken both in day and night.

1) Image(1)

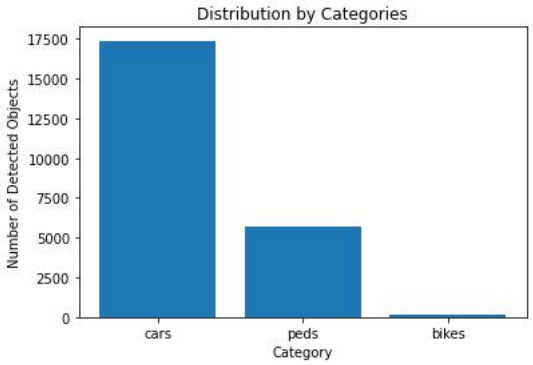
2) Image(2)



3) Image(6)



Cyclists are not as commonly observed as vehicles and pedestrians, which is confirmed by the bar graph below:



Cross Validation

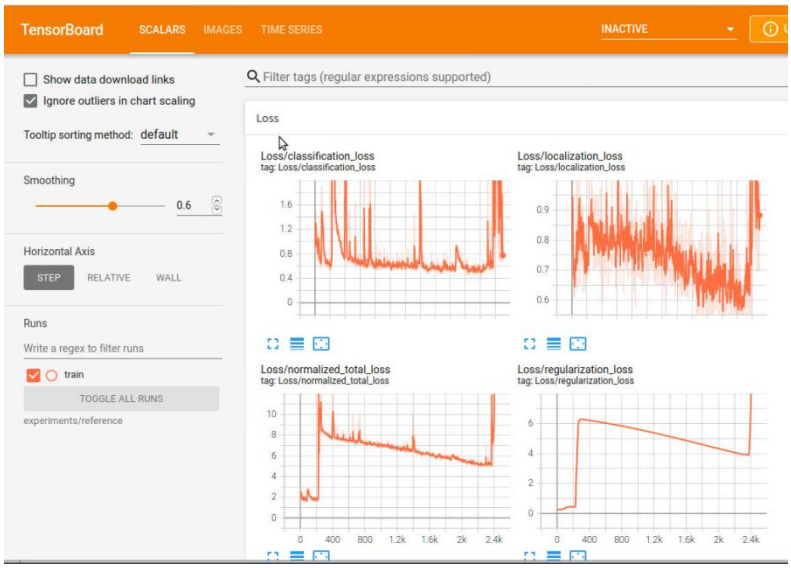
The provided Waymo dataset has a total of 100 tfrecord files. The percentages used to split the dataset into training, validation and testing sets are 80%, 15%, and 5%, respectively.

There is no fixed rule to decide the set ratios. It depends on the dataset. Each set should be representative of the data (e.g., has different light conditions) and the selected ratio should give optimal model performance. A very recent study done by Nguyen evaluated the performance of ML models for different training over test ratios: 10 : 90, 20 : 80, 30 : 70, 40 : 60, 50 : 50, 60 : 40, 70 : 30, 80 : 20, and 90 : 10 train/test split (1). The results showed that the training/testing ratio of 70/30 was the most suitable one for training and validating the ML models. For this project, I started with 70:20:10 train/validation/test ratio. But the initial runs were unable to get valid results for training. So I increased the training percentage to 80 and lowered the percentage for validation and testing accordingly. However, I later found a more dominant problem is the number of steps as 2500 is not sufficient.

Training

Reference Experiment

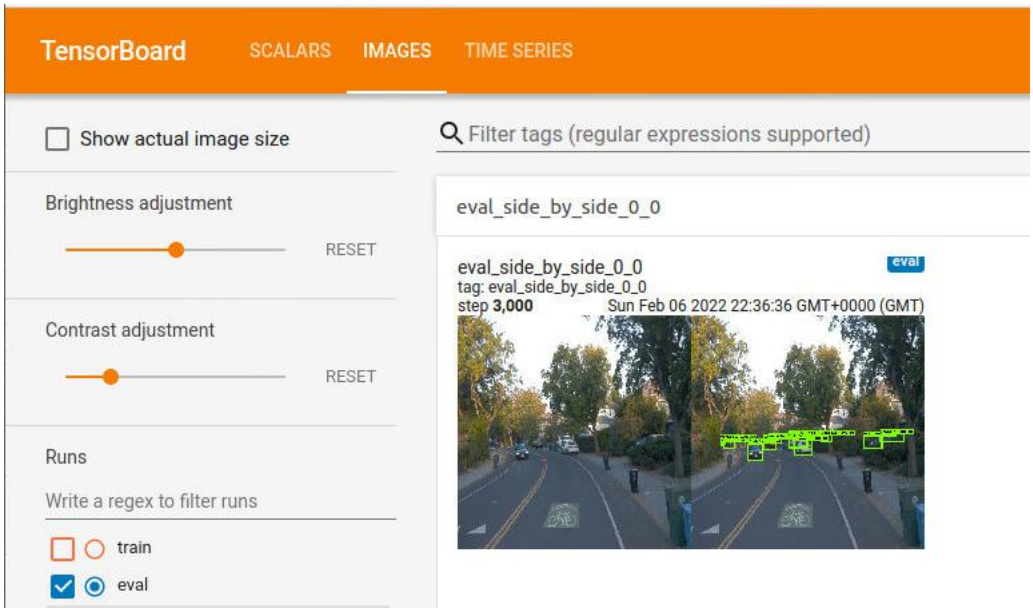
The reference experiment was conducted using the config file generated by directly running edit_config.py. The total_steps and warmup_steps were set as 2500 and 200. The training loss graph is as below:



The orange line represents the training loss and the total loss for training sets appear to decrease but is very unstable and doesn't seem to converge to an acceptable level. The spike toward the end of the graph is due to a broken run. When I restarted the run, it started over from ckpt-1 and resulted in a weird data point. I went back to check the validation run using a previously stored check point file but it didn't result in anything valuable. The mAP metrics are almost 0, as shown below:

```
INFO:tensorflow:     + DetectionBoxes_Precision/mAP@.75IOU: 0.000322
I0206 22:37:52.227032 140087091513088 model_lib_v2.py:991]  + DetectionBoxes_Precision/mAP@.75IOU: 0.000322
INFO:tensorflow:     + DetectionBoxes_Precision/mAP (small): 0.000540
I0206 22:37:52.229106 140087091513088 model_lib_v2.py:991]  + DetectionBoxes_Precision/mAP (small): 0.000540
INFO:tensorflow:     + DetectionBoxes_Precision/mAP (medium): 0.009258
I0206 22:37:52.231076 140087091513088 model_lib_v2.py:991]  + DetectionBoxes_Precision/mAP (medium): 0.009258
INFO:tensorflow:     + DetectionBoxes_Precision/mAP (large): 0.003296
I0206 22:37:52.232773 140087091513088 model_lib_v2.py:991]  + DetectionBoxes_Precision/mAP (large): 0.003296
INFO:tensorflow:     + DetectionBoxes_Recall/AR@1: 0.000063
I0206 22:37:52.235325 140087091513088 model_lib_v2.py:991]  + DetectionBoxes_Recall/AR@1: 0.000063
INFO:tensorflow:     + DetectionBoxes_Recall/AR@10: 0.002483
I0206 22:37:52.237853 140087091513088 model_lib_v2.py:991]  + DetectionBoxes_Recall/AR@10: 0.002483
INFO:tensorflow:     + DetectionBoxes_Recall/AR@100: 0.022417
I0206 22:37:52.240222 140087091513088 model_lib_v2.py:991]  + DetectionBoxes_Recall/AR@100: 0.022417
INFO:tensorflow:     + DetectionBoxes_Recall/AR@100 (small): 0.008393
I0206 22:37:52.242295 140087091513088 model_lib_v2.py:991]  + DetectionBoxes_Recall/AR@100 (small): 0.008393
INFO:tensorflow:     + DetectionBoxes_Recall/AR@100 (medium): 0.054906
I0206 22:37:52.244345 140087091513088 model_lib_v2.py:991]  + DetectionBoxes_Recall/AR@100 (medium): 0.054906
INFO:tensorflow:     + DetectionBoxes_Recall/AR@100 (large): 0.131818
I0206 22:37:52.246452 140087091513088 model_lib_v2.py:991]  + DetectionBoxes_Recall/AR@100 (large): 0.131818
INFO:tensorflow:     + Loss/localization_loss: 0.722937
I0206 22:37:52.249063 140087091513088 model_lib_v2.py:991]  + Loss/localization_loss: 0.722937
INFO:tensorflow:     + Loss/classification_loss: 0.741996
I0206 22:37:52.251238 140087091513088 model_lib_v2.py:991]  + Loss/classification_loss: 0.741996
INFO:tensorflow:     + Loss/regularization_loss: 1.722757
I0206 22:37:52.253165 140087091513088 model_lib_v2.py:991]  + Loss/regularization_loss: 1.722757
INFO:tensorflow:     + Loss/total_loss: 3.187690
I0206 22:37:52.255194 140087091513088 model_lib_v2.py:991]  + Loss/total_loss: 3.187690
```

This is consistent with the resulting image as show below. Even when the step gets to 3000, the image on the left is still showing 0 box, indicating the model is not well trained.



I also did a test run - using the model to generate the animation for a night-time condition (i.e., segment-12200383401366682847_2552_140_2572_140_with_camera_labels.tfrecord).
The animation shows no detected vehicles at all for the dark environment.

Improvements Over Reference

Based on the findings from the reference experiment. The problems need to solve include: 1) the

total loss for training is not converged after 3000 steps, 2) the model didn't work under dark environment.

To address the problem 1): the batch_size was increased from 2 to 4, and the total steps total_steps and warmup_steps were increased to 25,000 and 2,000. (Please note the results presented here is for steps ended at 12,000 because of an error indicating the resources and spaces on the workspace are not sufficient to run more steps. Also, the batch_size can't be further increased due to similar errors.)

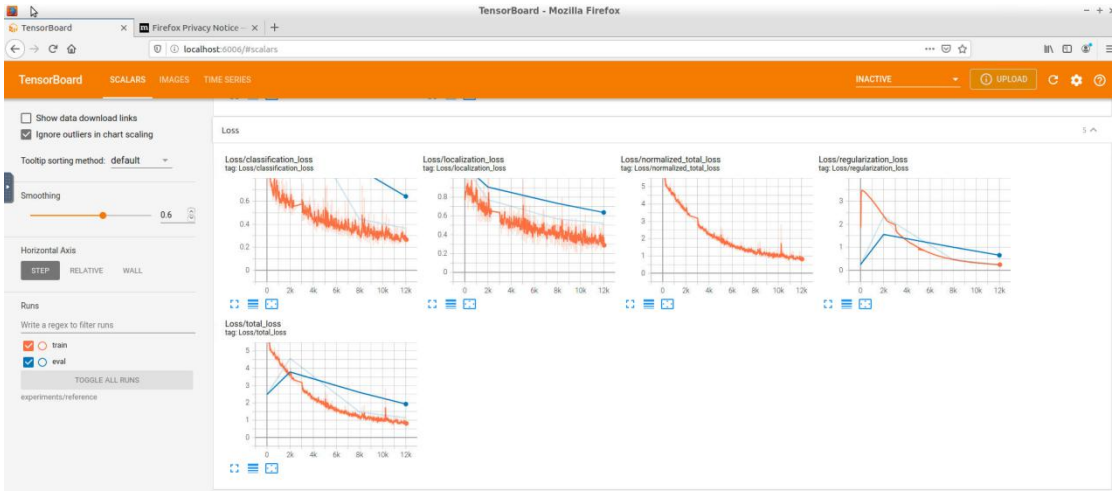
To address the problem 2), additional augmentation options were applied to the model to improve the performance. Two augmentations were added in the config file: random_adjust_brightness and random_adjust _contrast.

1. **random_adjust_brightness: apply default max_delta = 0.2**
2. **random_adjust _contrast: apply default min_delta = 0.8, max_delta = 1.25**

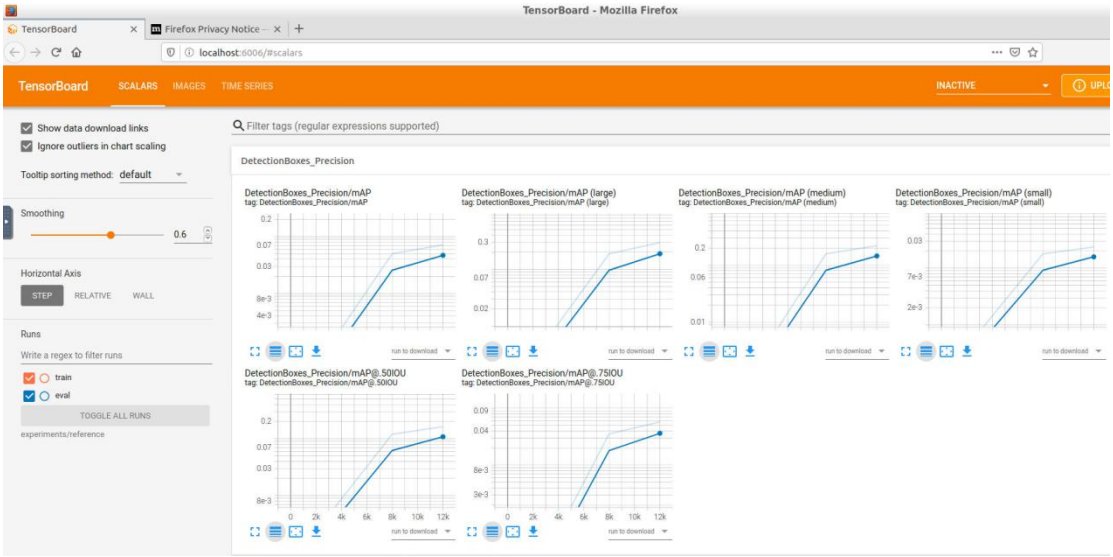
Due to the limited constraint, only the training and validation for the improved CNN model are presented in the figure below. The training loss is shown as an orange line and the validation loss is as an blue line. The total loss was lowered from 3.7 at step 2k to 0.8 at step 12k. This confirmed that the reference experiment has a total_step of 2.5k is too low.

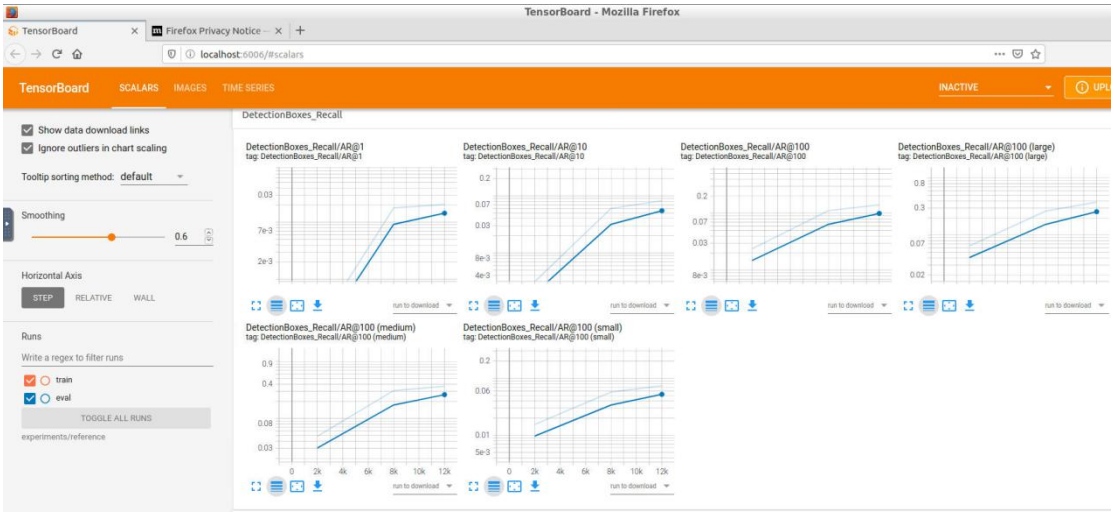
Due to the limited resources in the workspace, only four checkpoints can be included for the validation curve below.

Images of the improved model:



The resulting precision and recall charts of the improved model are shown below:





From the above graphs, the improved model is shown as dark blue lines. The precision of the improved model increased from 0.0 to 0.07 at step 12k, measured as mAP. The improved model increased the both precision and recall measures compared to reference model.

The same testing was performed and the improved model is able to detect vehicles in night-time environment. However, when it's very dark, the model can only detect the wheels of vehicles because those reflect the front lights and are brighter than other parts of the vehicles. Also, the model can't detect the vehicles from opposing vehicles with front light on.

In addition, a day-time condition was also tested and the improved model can detect vehicles in its surrounding but can't capture the vehicle with with large distance from it.

Reference

1. Quang Hung Nguyen et., al. Influence of Data Splitting on Performance of Machine Learning Models in Prediction of Shear Strength of Soil. Artificial Intelligence for Civil Engineering. Mathematical Problems in engineering, special issue, volume 2021.
https://www.hindawi.com/journals/mpe/2021/4832864/