Project: Object Detection in an Urban Environment

Project Overview:

In this project, the skills gained in the course are applied to create a convolutional neural network to detect and classify objects using data from Waymo. Data provided are images of urban environments containing annotated cyclists, pedestrians and vehicles. First, an extensive data analysis (EDA) is performed including the computation of label distributions, display of sample images, and checking for object occlusions. This analysis is used to decide what augmentations are meaningful for this project. Then, a neural network is trained to detect and classify objects. TensorBoard is used to monitor the training. Finally, different hyperparameters are used to improve the model's performance.

Set up:

Data

For this project, we will be using data from the Waymo Open dataset. We have already provided the data required to finish this project in the workspace, so you don't need to download it separately. However, if you are still interested in downloading the data locally, you can get it from the Google Cloud Bucket

Structure

Data:

The data you will use for training, validation and testing is organized in the /home/workspace/data/ directory as follows: train: contain the training data val: contain the validation data test - contains 10 files to test your model and create inference videos.

Experiments:

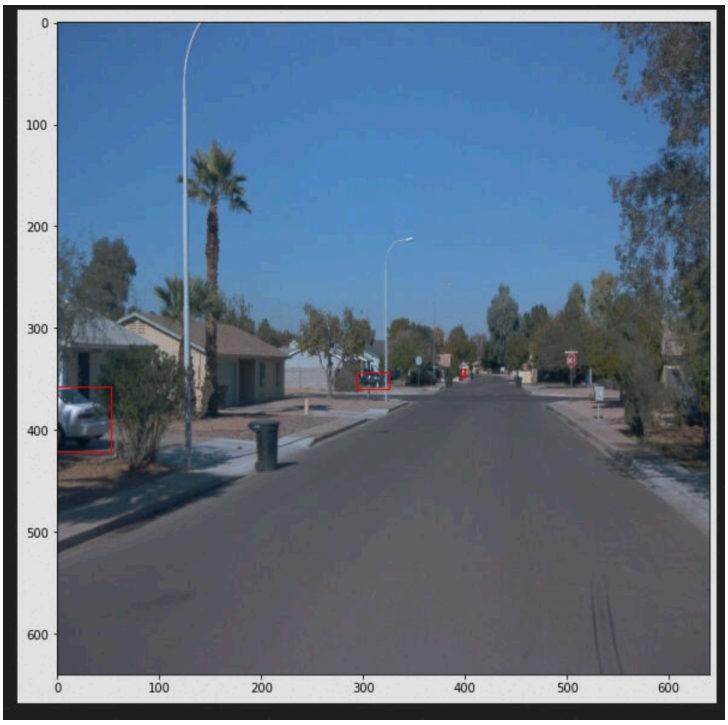
The /home/workspace/experiments folder is organized as follow:

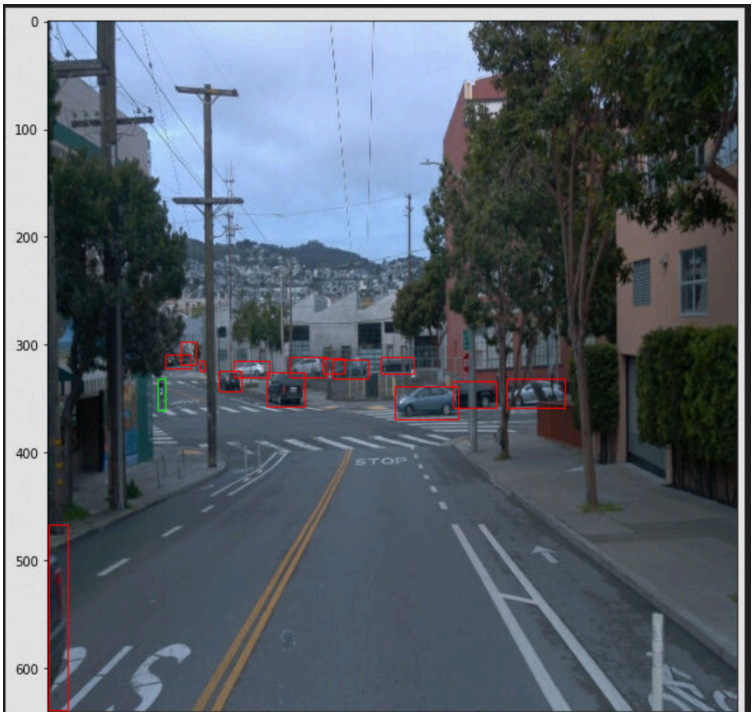
- pretrained_model reference - reference training with the unchanged config file
- exporter_main_v2.py - to create an inference model
- model_main_tf2.py - to launch training
- experiment0 - create a new folder for each experiment you run
- experiment1 - create a new folder for each experiment you run
- label_map.pbtxt

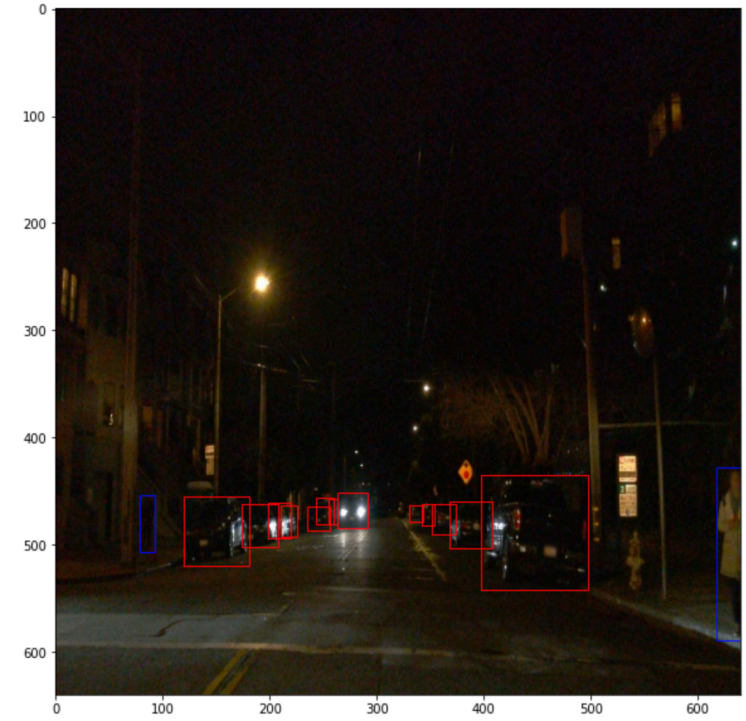
Dataset:

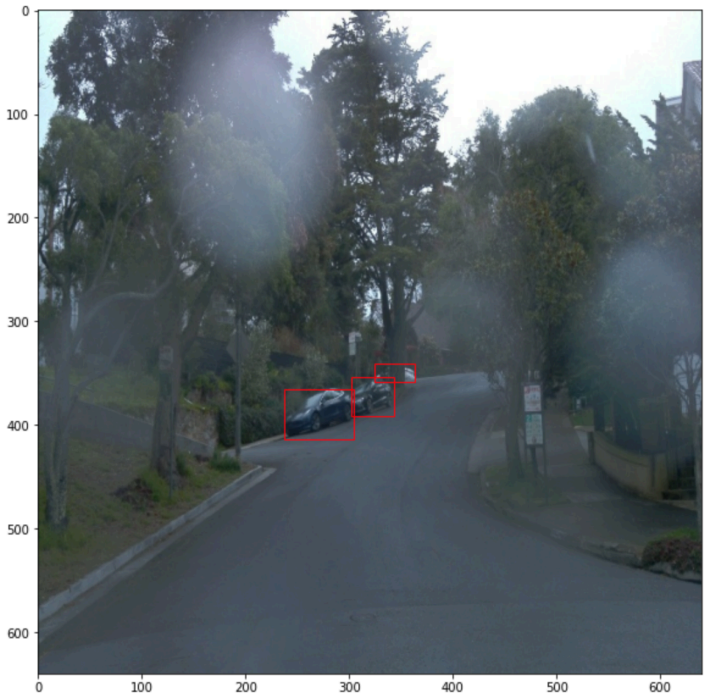
Dataset Analysis:

You should use the data already present in /home/workspace/data/ directory to explore the dataset! This is the most important task of any machine learning project. Implement the display_images function in the Exploratory Data Analysis notebook. This should be very similar to the function you created during the course. The output of this function looks like the image below:



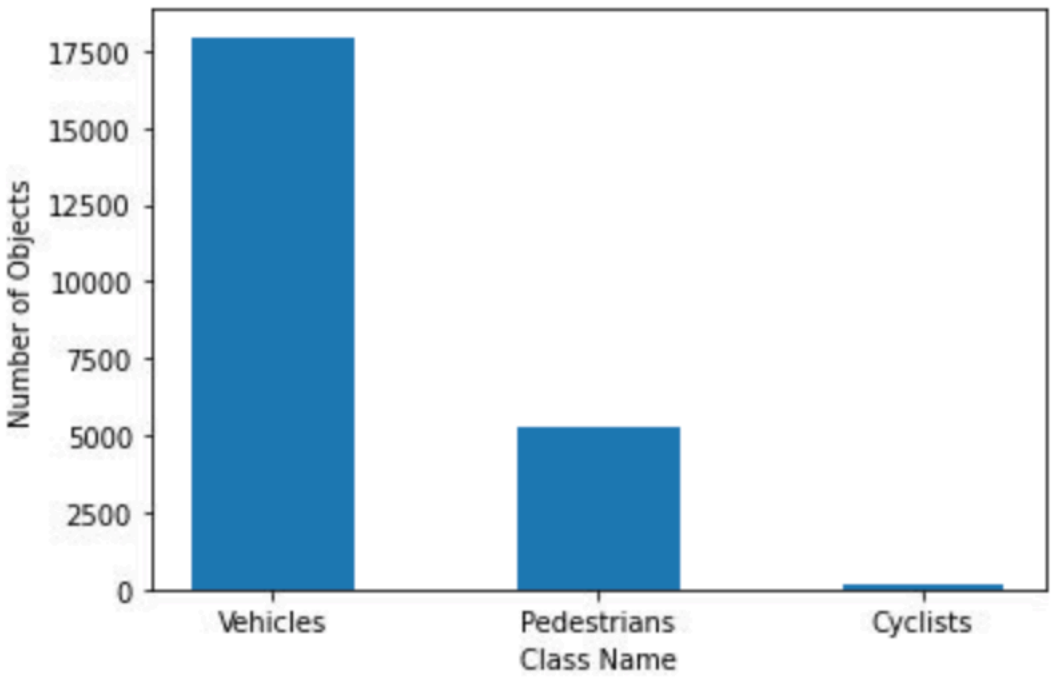

0
100
200
300
400
500
600
STOP


0
100
200
300
400
500
600
0
100
200
300
400
500
600



It can be observed from the above pictures that vehicles dominate the distribution of traffic participants (pedestrians and cyclists). So, the our model later will be able to predict vehicles quite well compared to other participants. Also, the images represent various environmental conditions like darkness, sunny day, foggy camera lens or droplets due to rain. All these will influences have to be taken into consideration while evaluating the model. Later, augmentations will be used to artificially induce some of these conditions to increase the robustness/generality of the model.

Additional EDA: The number of vehicles, pedestrians and cyclists in the “training”data have been plotted below. As can be seen, most of the identified objects are vehicles, followed by pedestrians and cyclists.



Cross validation:

The cross validation strategy used here is to randomly split the dataset into three parts named as training, testing and evaluation. Roughly 80% of the files are in training, 10% in testing and 10% in validation. The training set contains the most files because it is needed to train the model well on varied types of scenarios. As mentioned in "Additional EDA", cyclists are the least number here and hence most susceptible to mistakes during identification as the number of training points is not as high as the vehicle set.

Training:

Now we will perform the training. As explained during the course, the Tf Object Detection API relies on config files. The config that we will use for this project is pipeline.config, which is the config for a SSD Resnet 50 640x640 model.

First, let's download the pretrained model and move it to /home/workspace/experiments/pretrained_model/. Follow the steps below:

- cd /home/workspace/experiments/pretrained_model/ wget http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz

- tar -xvzf ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz

- rm -rf ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz

We need to edit the config files to change the location of the training and validation files, as well as the location of the label_map file, pretrained weights. We also need to adjust the batch size. To do so, predefined scripts are used as follows:

- cd /home/workspace/

- python edit_config.py --train_dir /home/workspace/data/train/ --eval_dir /home/workspace/data/val/ --batch_size 2 --checkpoint /home/workspace/experiments/pretrained_model/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckpt-0 --label_map /home/workspace/experiments/label_map.pbtxt

A new config file called pipeline_new.config will be created in the /home/workspace/ directory. Move this file to the /home/workspace/experiments/reference/ directory.

Reference Experiment:

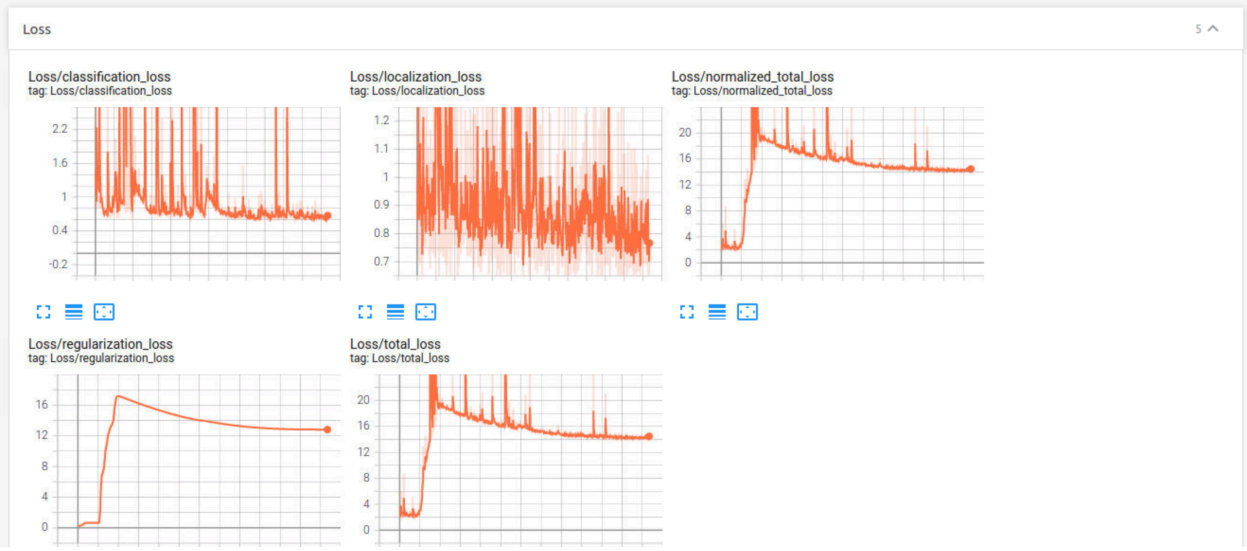
The training process is launched as follows:

python experiments/model_main_tf2.py --model_dir=experiments/reference/ —pipeline_config_path=experiments/reference/pipeline_new.config

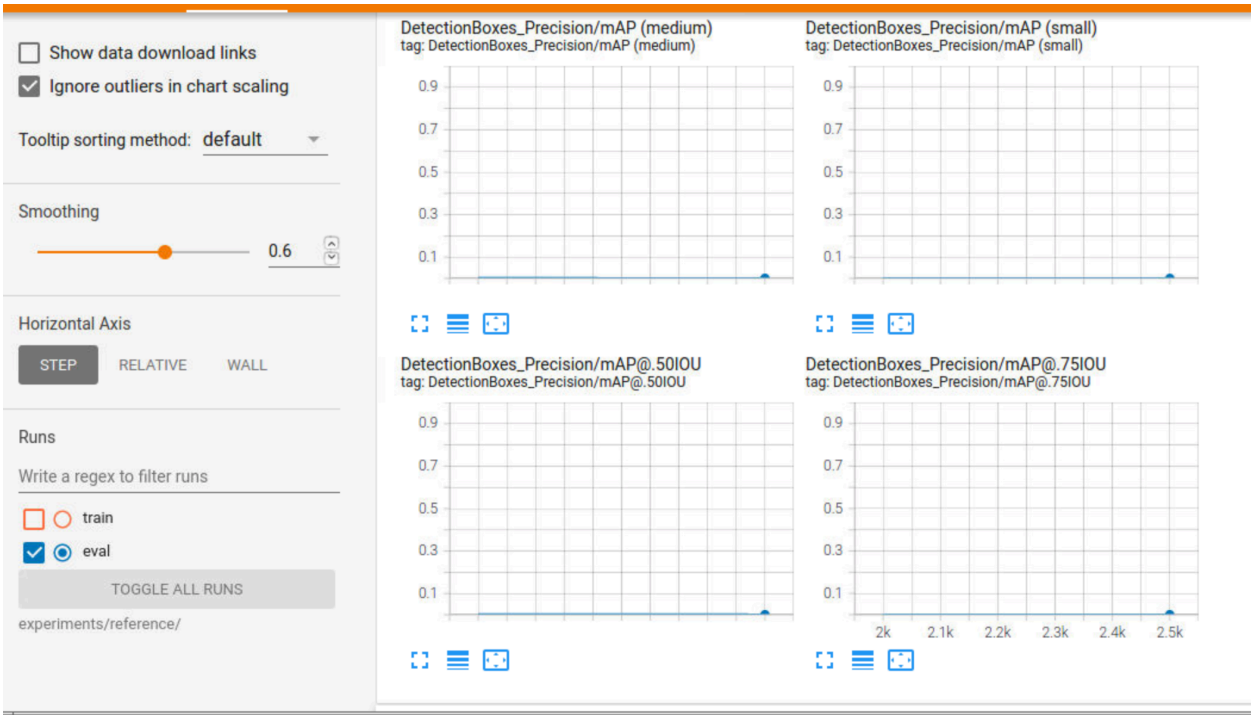
and the Tensorboard is used to monitor the progress as follows:

python -m tensorboard.main --logdir experiments/reference/

The results of the first trial are as follows (after 2500 iterations):



As you can observe, even after 2500 iterations, the total loss is around 14, which is quite high. This indicates a possible overfitting. Also, the loss starts to show static plateauing. The results were evaluated by calculating the mean Average Precision (mAP). As we have a high loss value, the model is not good at predictions. This is confirmed by the pretty low value of mAP, as below:



Improve on the reference:

To improve upon this, certain augmentation options were used in the pipeline_new.config file. The options used are:

- random_horizontal_flip
- random_rgb_to_gray
- random_adjust_contrast
- random_adjust_brightness

With the augmentation options, the following result could be achieved (original total loss: around 14; new total loss: 6). Although the loss has reduced, there is still potential

for some improvement through further experimentation. But, if even that does not help, more data may be needed to fit the model well.