Object Detection in an Urban Environment

Data

For this project, we will be using data from the [Waymo Open dataset](https://waymo.com/open/).

[OPTIONAL] - The files can be downloaded directly from the website as tar files or from the [Google Cloud Bucket](https://console.cloud.google.com/storage/browser/waymo_open_dataset_v_1_2_0_individual_files/) as individual tf records. We have already provided the data required to finish this project in the workspace, so you don't need to download it separately.

Structure

Data

The data you will use for training, validation and testing is organized as follow:
```
/home/workspace/data/waymo
        - training_and_validation - contains 97 files to train and validate your models
   - train: contain the train data (empty to start)
   - val: contain the val data (empty to start)
   - test - contains 3 files to test your model and create inference videos
```
The `training_and_validation` folder contains file that have been downsampled: we have selected one every 10 frames from 10 fps videos. The `testing` folder contains frames from the 10 fps video without downsampling.
```
You will split this `training_and_validation` data into `train`, and `val` sets by completing and executing the `create_splits.py` file.


### Experiments
The experiments folder will be organized as follow:
```
experiments/
   - pretrained_model/
   - exporter_main_v2.py - to create an inference model
   - model_main_tf2.py - to launch training
   - reference/ - reference training with the unchanged config file
   - experiment0/ - create a new folder for each experiment you run
   - experiment1/ - create a new folder for each experiment you run
   - experiment2/ - create a new folder for each experiment you run
   - label_map.pbtxt
   ...
```
```

Prerequisites

Local Setup

For local setup if you have your own Nvidia GPU, you can use the provided Dockerfile and requirements in the [build directory](./build).

Follow [the README therein](./build/README.md) to create a docker container and install all prerequisites.

Download and process the data

Note: ”If you are using the classroom workspace, we have already completed the steps in the section for you. You can find the downloaded and processed files within the `/home/workspace/data/preprocessed_data/` directory. Check this out then proceed to the **Exploratory Data Analysis** part.

The first goal of this project is to download the data from the Waymo's Google Cloud bucket to your local machine. For this project, we only need a subset of the data provided (for example, we do not need to use the Lidar data). Therefore, we are going to download and trim immediately each file. In `download_process.py`, you can view the `create_tf_example` function, which will perform this processing. This function takes the components of a Waymo Tf record and saves them in the Tf Object Detection api format. An example of such function is described [here](https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#create-tensorflow-records). We are already providing the `label_map.pbtxt` file.

You can run the script using the following command:
```
python download_process.py --data_dir {processed_file_location} --size {number of files you want to download}
```

You are downloading 100 files (unless you changed the `size` parameter) so be patient! Once the script is done, you can look inside your `data_dir` folder to see if the files have been downloaded and processed correctly.

Classroom Workspace

In the classroom workspace, every library and package should already be installed in your environment. You will NOT need to make use of `gcloud` to download the images.

Instructions

Exploratory Data Analysis

You should use the data already present in `/home/workspace/data/waymo` directory to explore the dataset! This is the most important task of any machine learning project. To do so, open the `Exploratory Data Analysis` notebook. In this notebook, your first task will be to implement a `display_instances` function to display images and annotations using `matplotlib`. This should be very similar to the function you created during the course. Once you are done, feel free to spend more time exploring the data and report your findings. Report anything relevant about the dataset in the writeup.

Keep in mind that you should refer to this analysis to create the different spits (training, testing and validation).

Create the training - validation splits
In the class, we talked about cross-validation and the importance of creating meaningful training and validation splits. For this project, you will have to create your own training and validation sets using the files located in `/home/workspace/data/waymo`. The `split` function in the `create_splits.py` file does the following:
* create three subfolders: `/home/workspace/data/train/`, `/home/workspace/data/val/`, and `/home/workspace/data/test/`
* split the tf records files between these three folders by symbolically linking the files from `/home/workspace/data/waymo/` to `/home/workspace/data/train/`, `/home/workspace/data/val/`, and `/home/workspace/data/test/`

Use the following command to run the script once your function is implemented:
```
python create_splits.py --data-dir /home/workspace/data
```

Edit the config file

Now you are ready for training. As we explain 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. You can learn more about the Single Shot Detector [here](https://arxiv.org/pdf/1512.02325.pdf).

First, let's download the [pretrained model](http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz) and move it to `/home/workspace/experiments/pretrained_model/`.

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, run the following:
```
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 has been created, `pipeline_new.config`.

Training

You will now launch your very first experiment with the Tensorflow object detection API. Move the `pipeline_new.config` to the `/home/workspace/experiments/reference` folder. Now launch the training process:
* a training process:
```
```

python experiments/model_main_tf2.py --model_dir=experiments/reference/ --
pipeline_config_path=experiments/reference/pipeline_new.config
```
Once the training is finished, launch the evaluation process:
* an evaluation process:
```
python experiments/model_main_tf2.py --model_dir=experiments/reference/ --
pipeline_config_path=experiments/reference/pipeline_new.config
--checkpoint_dir=experiments/reference/
```

**Note**: Both processes will display some Tensorflow warnings, which can be ignored. You may have to kill the evaluation script manually using
`CTRL+C`.

To monitor the training, you can launch a tensorboard instance by running `python -m tensorboard.main --logdir experiments/reference/`. You will report your findings in the writeup.

### Improve the performances

Most likely, this initial experiment did not yield optimal results. However, you can make multiple changes to the config file to improve this model. One obvious change consists in improving the data augmentation strategy. The
[`preprocessor.proto`](https://github.com/tensorflow/models/blob/master/research/object_detection/protos/preprocessor.proto) file contains the different data augmentation method available in the Tf Object Detection API. To help you visualize these augmentations, we are providing a notebook: `Explore augmentations.ipynb`. Using this notebook, try different data augmentation combinations and select the one you think is optimal for our dataset. Justify your choices in the writeup.

Keep in mind that the following are also available:
* experiment with the optimizer: type of optimizer, learning rate, scheduler etc
* experiment with the architecture. The Tf Object Detection API [model zoo](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md) offers many architectures. Keep in mind that the `pipeline.config` file is unique for each architecture and you will have to edit it.

**Important:** If you are working on the workspace, your storage is limited. You may to delete the checkpoints files after each experiment. You should however keep the `tf.events` files located in the `train` and `eval` folder of your experiments. You can also keep the `saved_model` folder to create your videos.

### Creating an animation
#### Export the trained model
Modify the arguments of the following function to adjust it to your models:

```
python experiments/exporter_main_v2.py --input_type image_tensor --pipeline_config_path
experiments/reference/pipeline_new.config --trained_checkpoint_dir experiments/reference/ --
output_directory experiments/reference/exported/
```
```

This should create a new folder `experiments/reference/exported/saved_model`. You can read more about the Tensorflow SavedModel format [here](https://www.tensorflow.org/guide/saved_model).

Finally, you can create a video of your model's inferences for any tf record file. To do so, run the following command (modify it to your files):
```
python inference_video.py --labelmap_path label_map.pbtxt --model_path
experiments/reference/exported/saved_model --tf_record_path /data/waymo/testing/segment-
12200383401366682847_2552_140_2572_140_with_camera_labels.tfrecord --config_path
experiments/reference/pipeline_new.config --output_path animation.gif
```

Submission Template

Project overview

This is a repository containing the details of the project Udacity Self Driving Car, where we are using Tf object detection API for better detetion of objects such as cars,pedestrians, cyclists etc. This repository contains the details to download the tfrecord sample files from Cloud storage and then split them for training purposes on the object detection API. The dataset used for this purpose is [Waymo](https://waymo.com/open/) which can be downloaded from the [Google Cloud Storage Bucket]((https://console.cloud.google.com/storage/browser/waymo_open_dataset_v_1_2_0_individual_files/). In this case, we will be using tfrecord files which we will be modified into tf.Train.Example for the object detection api format. We will also be splitting the dataset into training, validation and testing sets using np.split in "create_splits.py" python program.

Set up

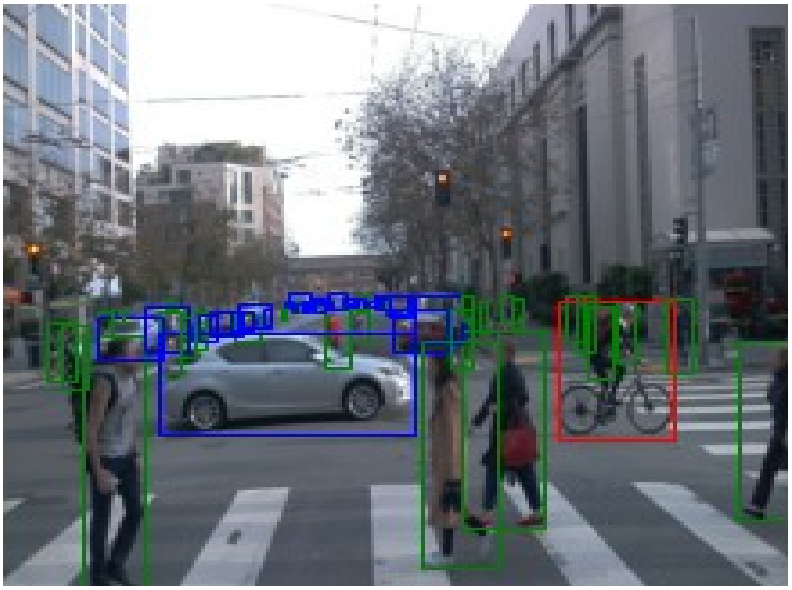
As mentioned in the project rubrics, GPU system should be enable while executingin Udacity workspace. In my case, I have used udacity workspace and udacity virtual Desktop.

- First the project files should be downloaded through git clone from [this repository](https://github.com/udacity/nd013-c1-vision-starter)

In case of using Udacity workspace, No need to install any extra packages or libraries and to gain this advantage , must enable GPU given by UDACITY.

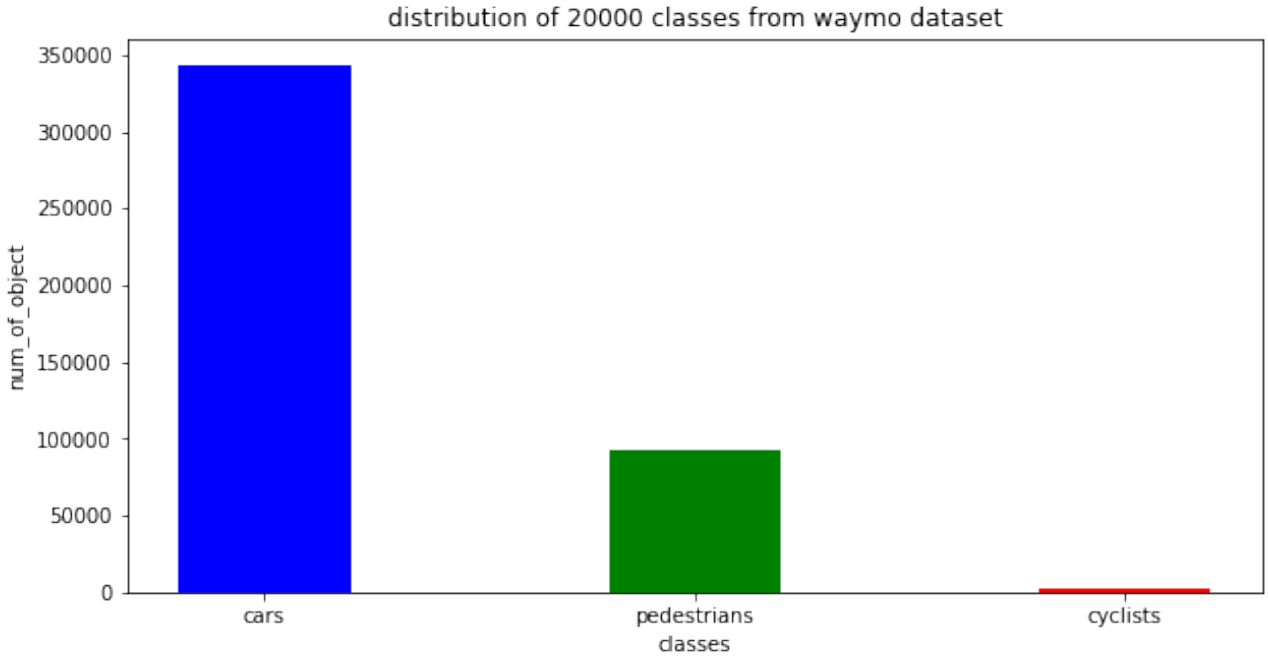
Dataset

In the dataset, we have to fit rectangular bounding boxes on the images with objects ,which includes pedestrians, cyclists and cars.Images are taken from different places, and different weather conditions and at different time of the day (day/night).The image set contains diverse set of images of which some are blurry, clear, light and some are dark. A sample image in little blurry and dark background is provided below:

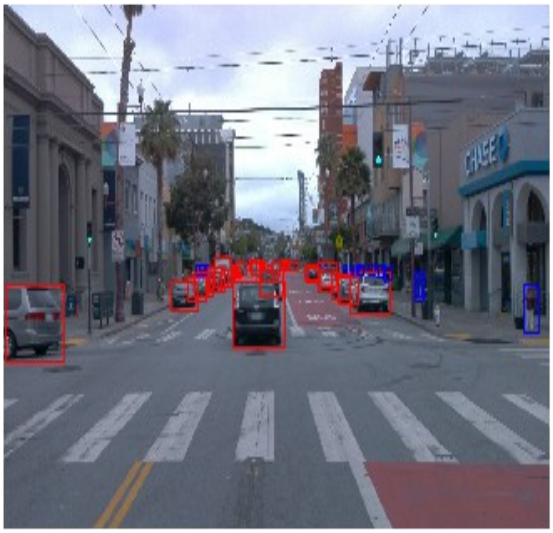


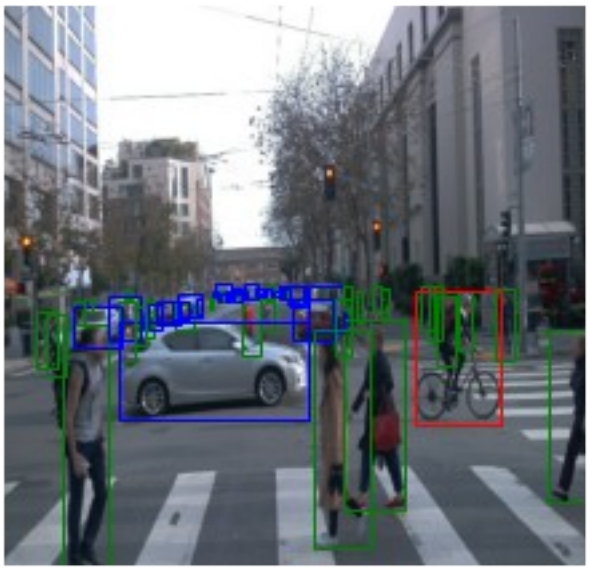
Dataset analysis

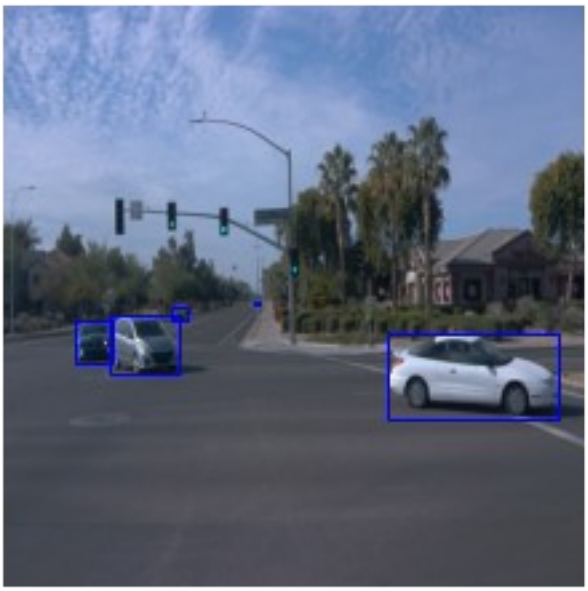
The dataset presents most of the labels being associated with cars and pedestrians with the sample size of cyclists being very small. The proportionate amount of the counts of the different labels are in "Exploratory Data Analysis.ipynb" file. There is a class imbalance which can be handled with oversampling techniques. A sample image of the proportional counts of the labels (cars,pedestrians,cyclists) are shown below:

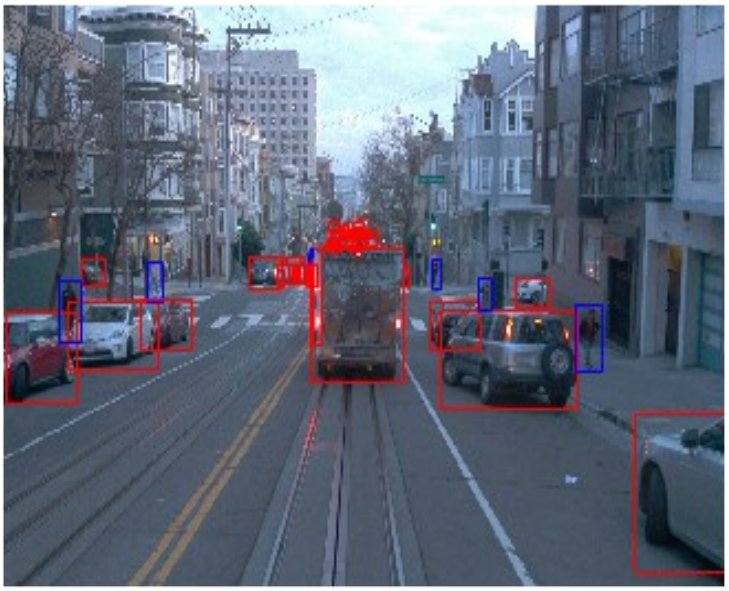


Images are taken in different environments(subway/highway/city) with different weather conditions(foggy/sunny) and different times of the day(day/night).The bounding boxes are red for the vehicles, green for the cyclists and blue for the pedestrians.









From above bar graph of the dataset shows that most images contain vehicles and pedestrians (majority vehicles), and very few sample images have cyclists in them. Bar plot for the distribution of classes (cars, pedestrians, and cyclists), over a collection of 20000 random images in the dataset. The analysis is updated in the "Exploratory Data Analysis.ipynb" notebook.

Cross validation

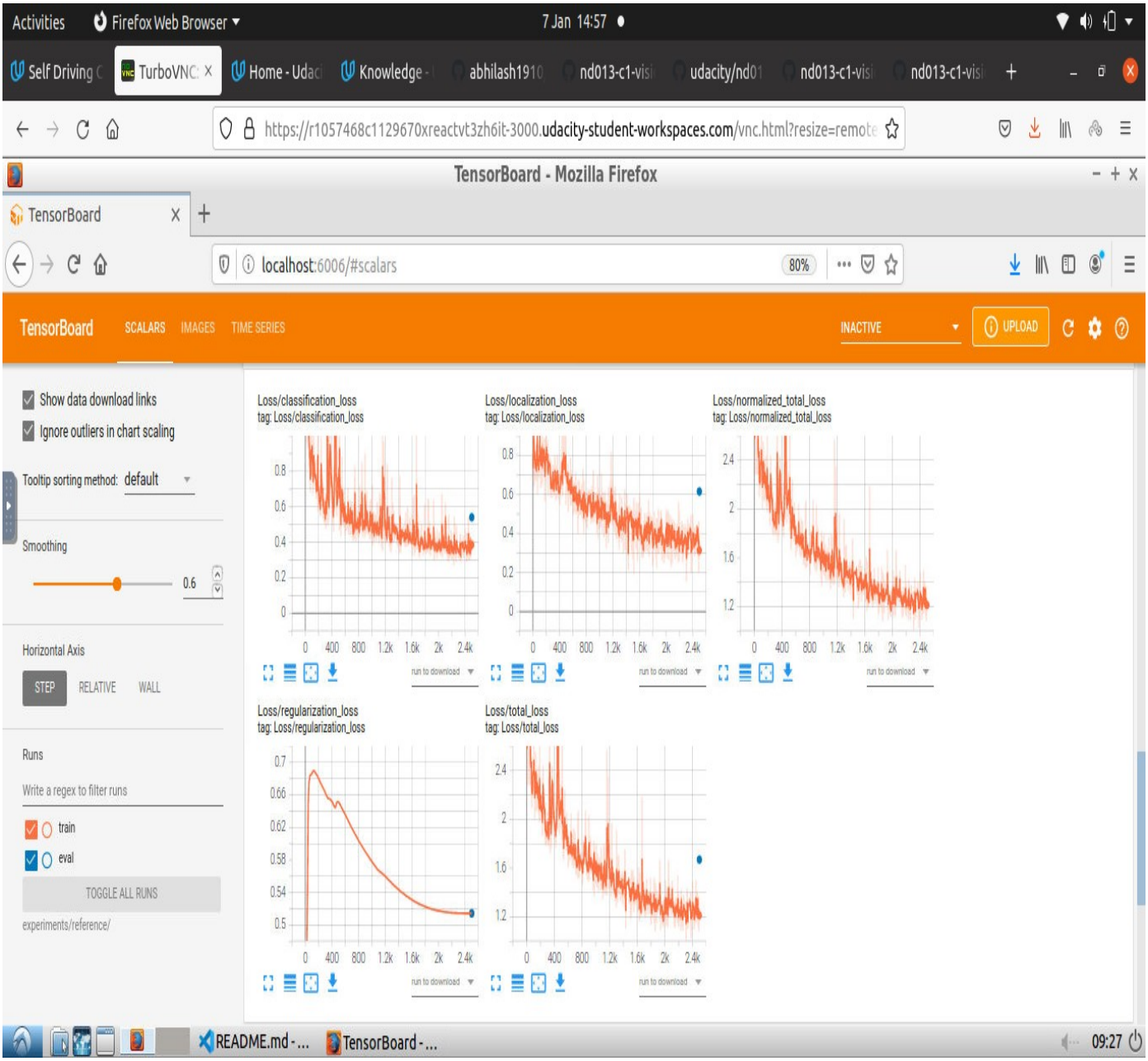
We are using 100 tfrecord files. We first shuffle the data randomly and then split into training,testing and validation sets. The reason for random shuffling is to reduce the class imbalance in each sample. The shuffling ensures approximately equal distribution of samples in the training,testing and validation datasets.

In this case, we are using 0.75 : 0.15 as the proportion of training and validation data since we are using only 100 tfrecord samples. This ensures that we have sufficient data for training as well as validation.We are using 10% (0.1) of the sample as the test set to check the error rate and if the model is overfitting.Ideally,overfitting should not be an issue since 75% is under training and rest 10% is for testing.

Training

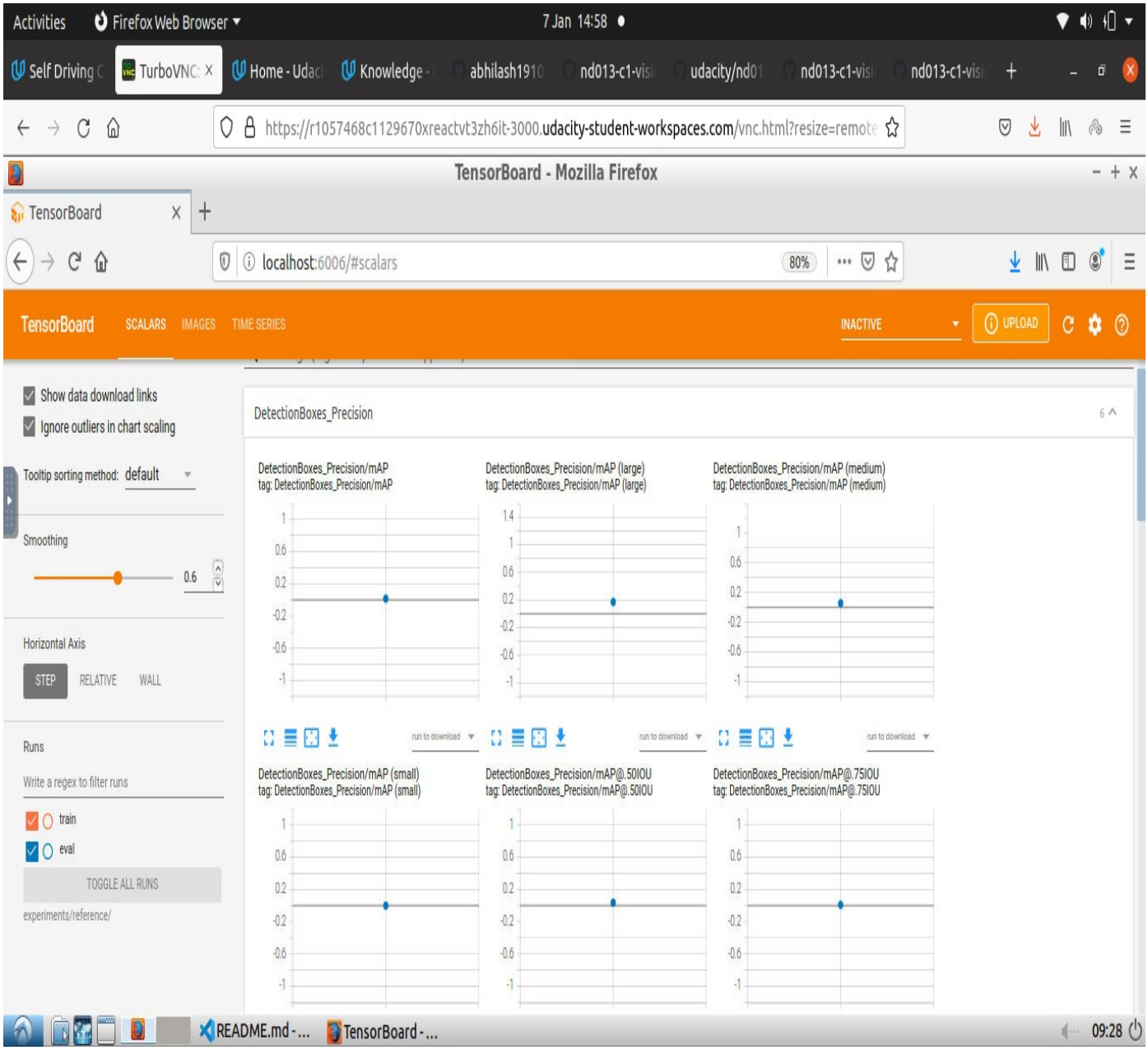
Reference experiment

The residual network model ([Resnet](http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz)) without augmentation , model loss is shown below:

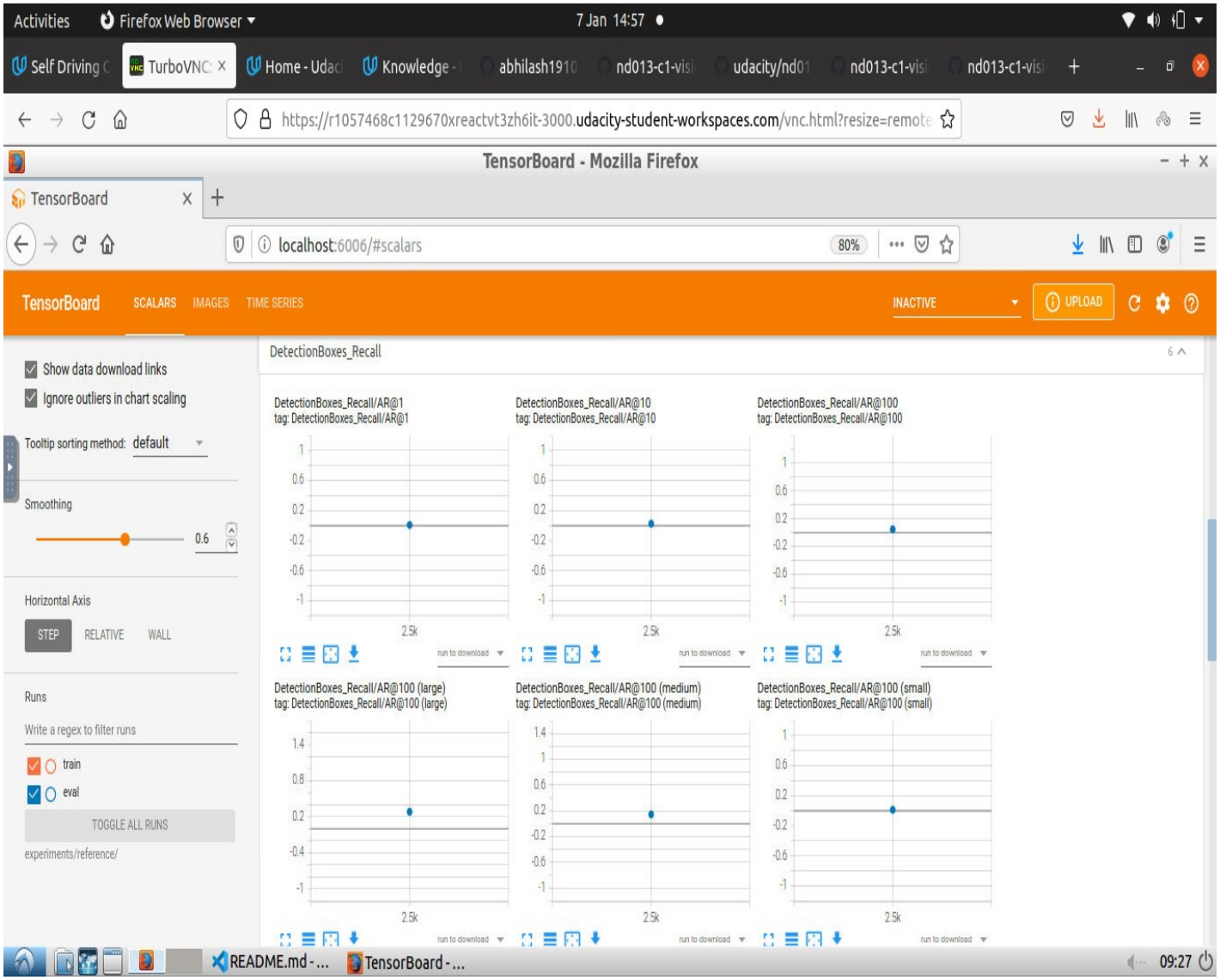


Actually , I don't understand this graph plotting ,what is going in above . However , with my little knowledge, I can say that the blue dot and red line seems doesnot diverses more .It mean model is little overfitting. Below there are precision and recall graph, these blue dotted in both precision and recall for boxes detection does show increment constantly but on slow .(Actually I do not understand why only one blue dot plotted in graph.)

Precision:


Activities
Firefox Web Browser
7 Jan 14:58
TensorBoard - Mozilla Firefox
TensorBoard
localhost:6006/#scalars
TensorBoard
SCALARS
IMAGES
TIME SERIES
INACTIVE
UPLOAD
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train
eval
TOGGLE ALL RUNS
experiments/reference/
DetectionBoxes_Precision
DetectionBoxes_Precision/mAP
tag: DetectionBoxes_Precision/mAP
DetectionBoxes_Precision/mAP (large)
tag: DetectionBoxes_Precision/mAP (large)
DetectionBoxes_Precision/mAP (medium)
tag: DetectionBoxes_Precision/mAP (medium)
DetectionBoxes_Precision/mAP (small)
tag: DetectionBoxes_Precision/mAP (small)
DetectionBoxes_Precision/mAP@.50IOU
tag: DetectionBoxes_Precision/mAP@.50IOU
DetectionBoxes_Precision/mAP@.75IOU
tag: DetectionBoxes_Precision/mAP@.75IOU
run to download
README.md - ...
TensorBoard - ...
09:28

Recall:



Improve on the reference

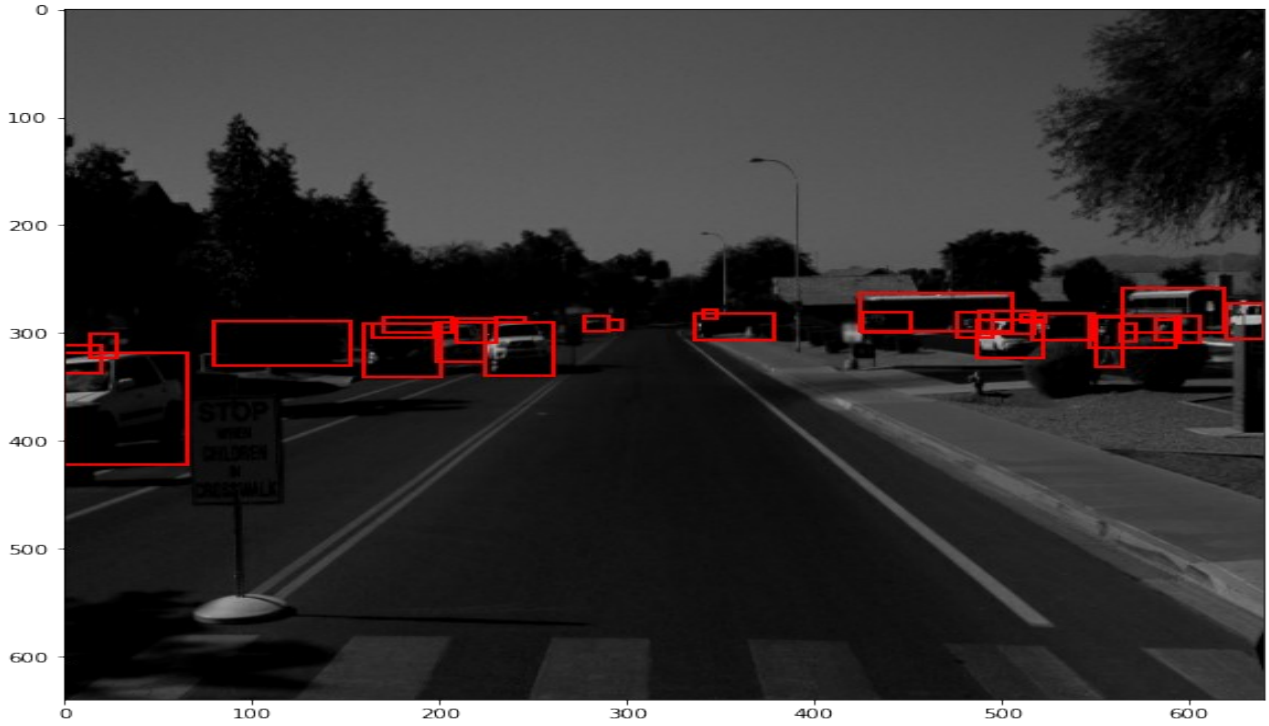
To improve on the model performance, the first step was to augment the images by converting them to grayscale with a probability of 0.3. After this, we have clamped the contrast values between 0.8 and 1.02such that more lighting datapoints are available for classification. A greater part of the images were a bit darker and increasing the brightness to 0.4 provided an even datapoint which could be better classified with the model.The pipeline changes are there in ```pipeline_new.config``` under results directory .

Augmentations applied:

- 0.3 probability of grayscale conversion
- brightness adjusted to 0.4
- contrast values between 0.8 and 1.2

Night(Darker) Images:

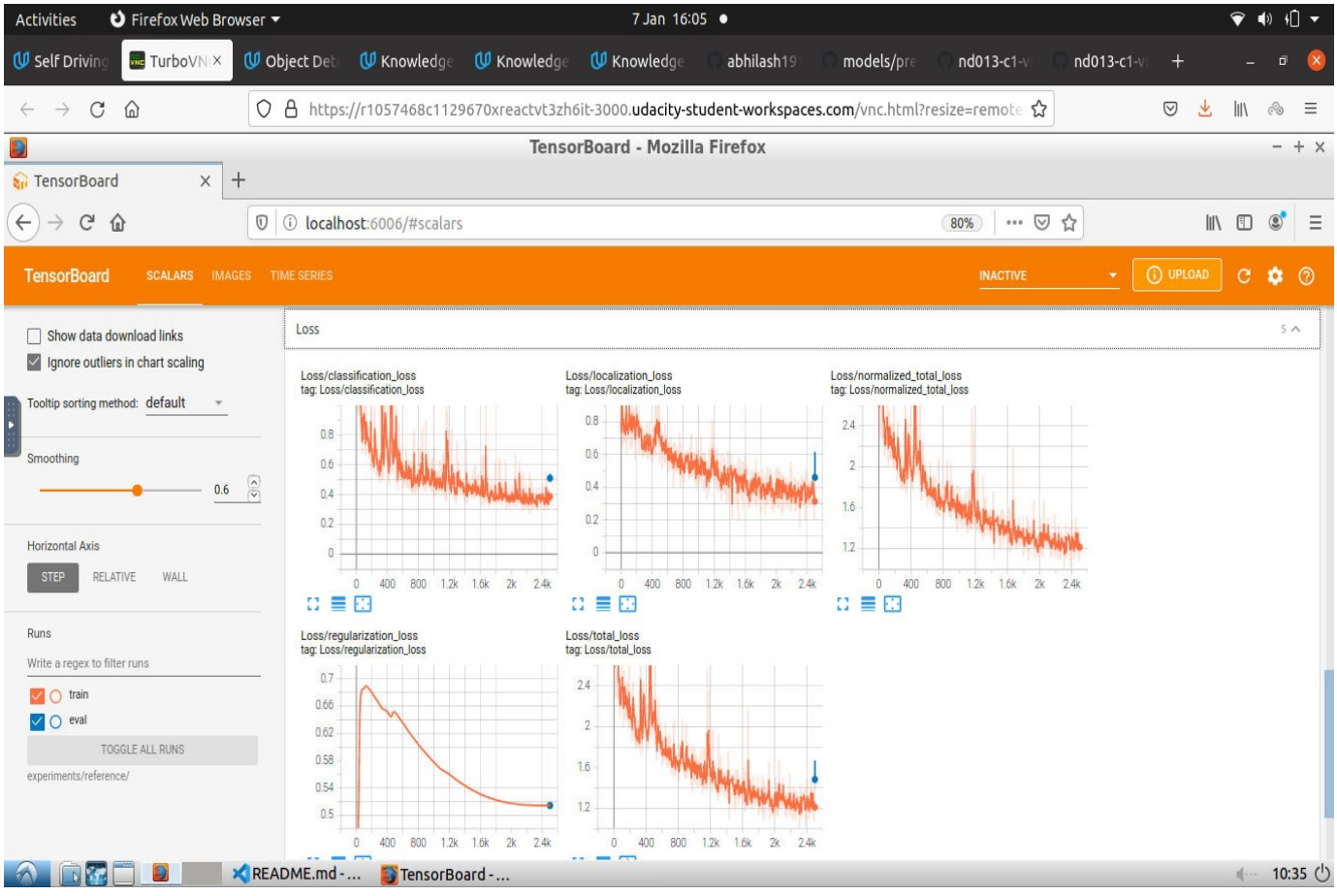
Greyscale Images:



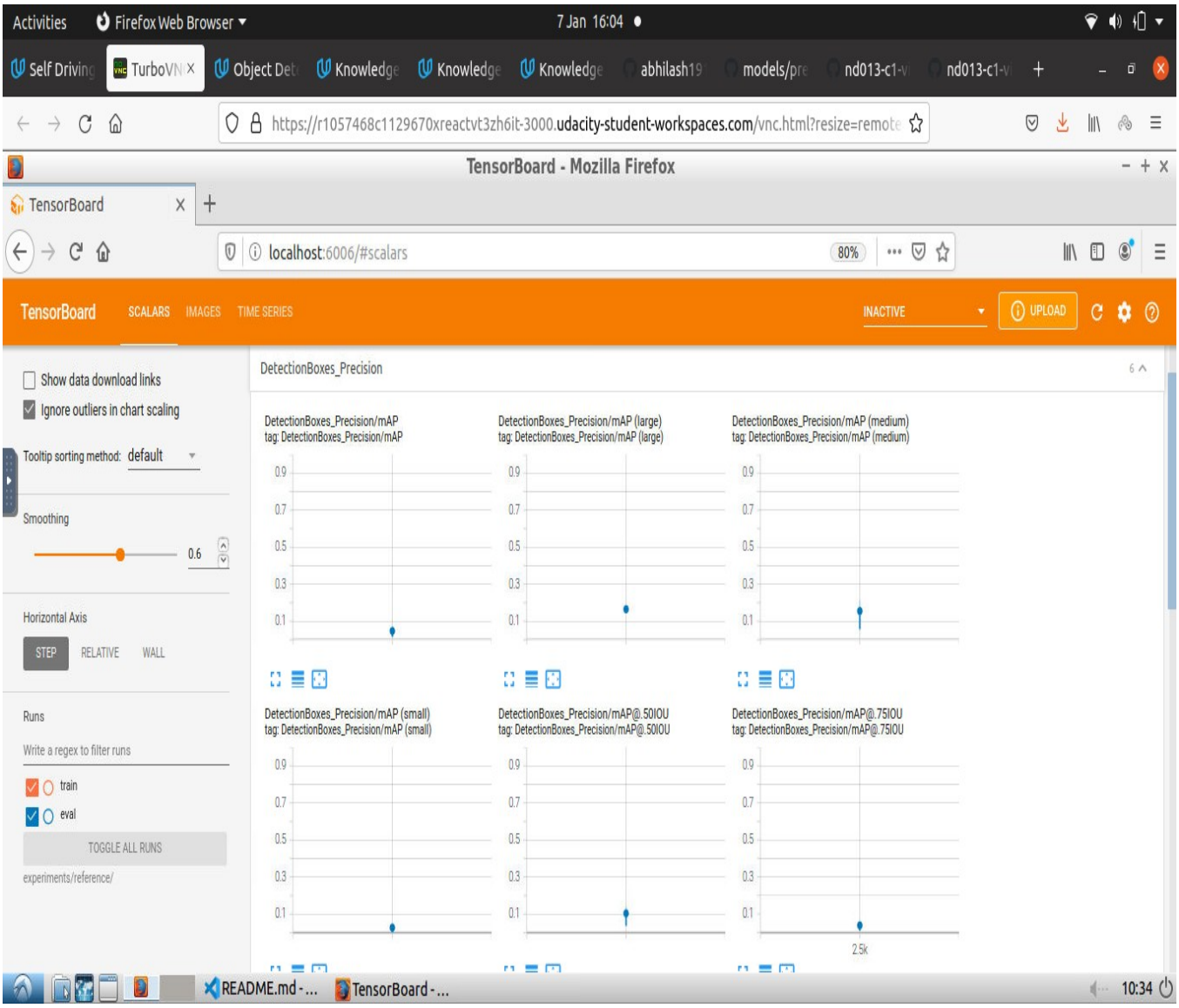
Brighter Image:

The details of the run can be found here : "Explore augmentations.ipynb"

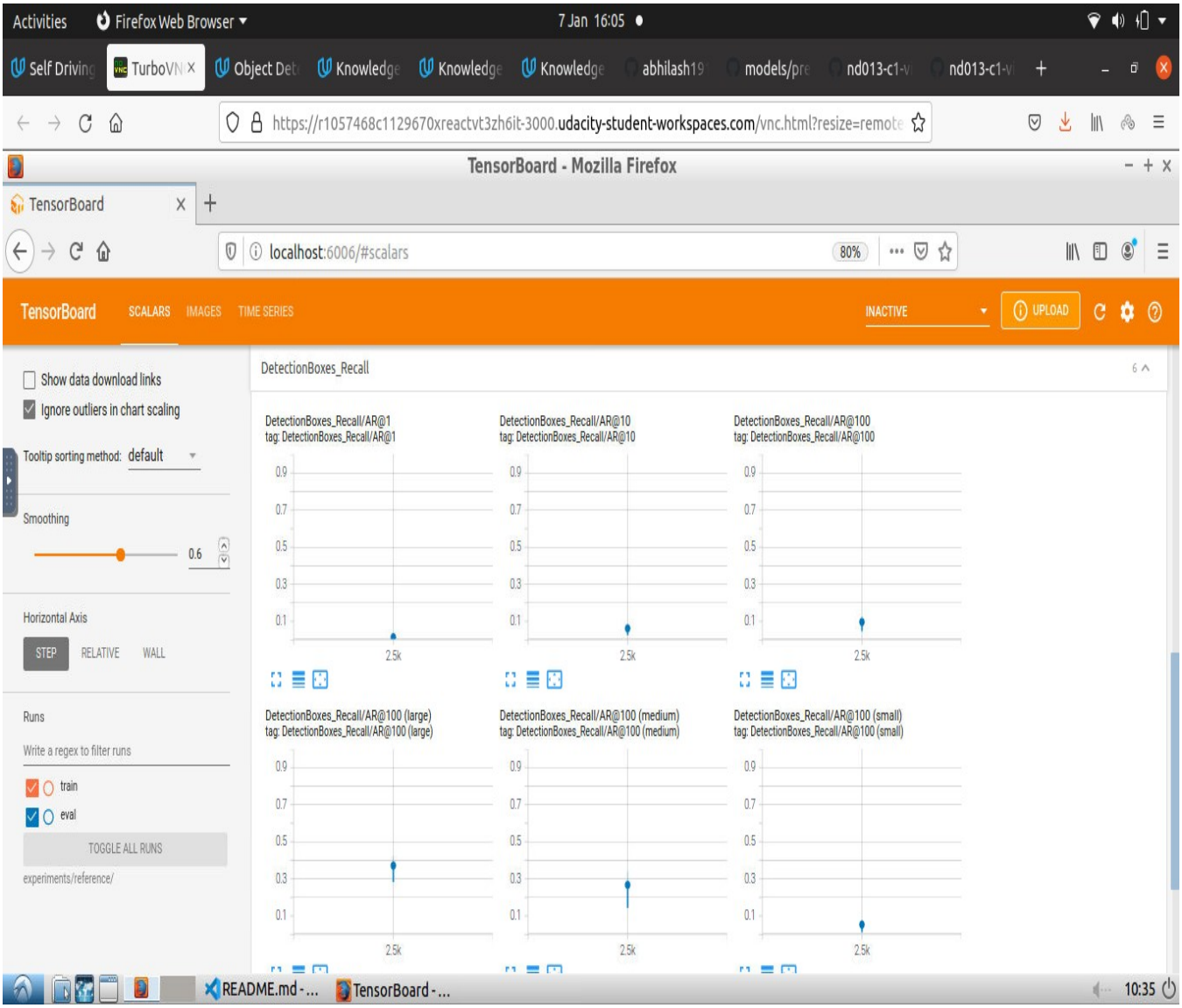
The model loss with augmentation :



Precision with Augmentation:



Recall with Augmentation:



The loss is lower than the previous loss (un-augmented model). This is an indication of better performance. There should be more samples of augmented datapoints such as
combining the contrast values with grayscale. Brightness can also be clamped within a limit instead of fixing it to 0.3
However the most important point is to add more samples of cyclists,pedestrians which are in a low quantity in the dataset. This is an inherent requirement since model biases play an important role in the loss curves and lesser the diversity in training samples, the lower will be the accuracy.

We have reduced overfitting to an extent with augmentation, however better classification results would be resulting from a more balanced dataset.