OBJECT DETECTION IN AN URBAN ENVIRONMENT

Project Overview

The objective of this assignment is to train a dataset for object detection in an urban environment using resnet.

Setup

I must say this has been one of the most frustrating experiences in my life. Something that could have taken hours ended up taking me days because of various technical issues and poorly written instructions.

The workspace kept going idle while I am working on it. Jupyter Notebook kept crashing both on Firefox and Chromium Browser. It got so bad that I had to figure out how to get this thing running on my desktop PC.

I had no experience with Docker and didn't even know what it was so, of course, I configured it wrong. Eventually, I did figure it out and got it working.

Dataset

The data was downloaded from the Waymo open dataset. Those files are huge. After 10 hours, I had only downloaded 32 files. I decided that was enough for what I needed to do.

I used those files to write the code in Exploratory Data Analysis, then copied them back to the provided workspace.

#Experiment

I started with the default config file and created the base reference. The model prediction was pretty bad with a total loss of around 3.2.

I must say that training from the desktop environment is a pain. You have to keep on clicking that box that pops up every few minutes asking if you want to continue using the desktop. In other words, you have to be a slave to it and sit there doing nothing for hours. Eventually, Although the instructions say to delete the training data leaving only the tf.events, that won't work. It will only leave you frustrated for hours. Because you don't have enough room in the workspace to leave the tfevents and complete a training session, you will end up running out of space and you won't be able to complete training. Ideally, we would need an extra 2gb of space to follow the instructions and complete the training.

Additionally, it's important that raining and eval are launched from outside the desktop environment in the cli that comes up after entereing the project workspace. If not, the workspace will keep going idle as you are working on the project.That information should be included in the project instructions. Furthermore, deleting files doesn't remove them from your workspace as they go in the trash. You cannot empty the trash in this environment the way you would do it with a regular box running Debian or Ubuntu. Don't bother reaching out for support on that subject. They will get back to you after a couple of days and won't provide any valuable information. I have found that the easiest way to delete the trash in the workspace is to install trash-cli (apt-get install trash-cli) and issue the trash-empty command. For local setup, an Nvidia card isn't enough. 10 series won't work with the latest drivers. Support for the 10 series was dropped after nvidia-driver-515. The current nvidia driver installed by default with the Docker script is nvidia-driver-516. It's a pain in the ass to downgrade to 510 or 515 especially if you are new to this environment. For things to go smoothly, you will need at least a 20 series since they have the required Cuda cores.I wish this had been included in the information about getting a local setup up and running.

Setting up locally is highly advisable in order to avoid the frustration of the frequent crashes of the workspace. Our time should be spent fine-tuning the training process instead of trying to fix environment issues. Also, if we do the work from within the workspace why do we have to submit through Github?

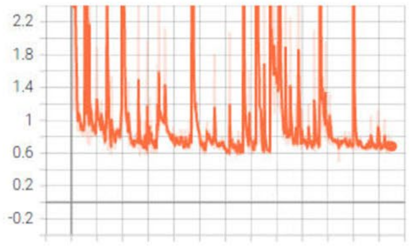
THE EXPERIMENTS

For the first experiment, I trained the model with the default settings provided with pipeline_new.config. With those settings, our normalized loss was very high at around 3.2 while the learning rate got worse as the training progressed as shown by the charts below.

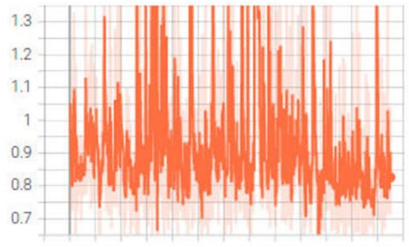
Loss

5

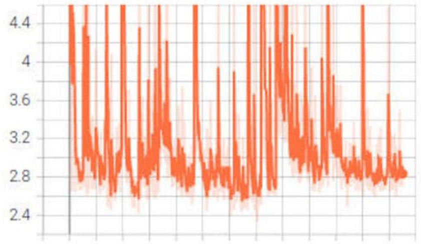
Loss/classification_loss
tag: Loss/classification_loss



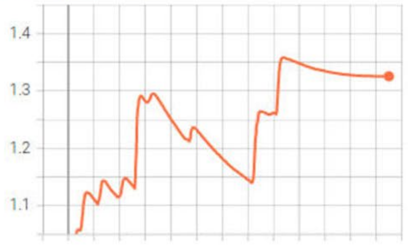
Loss/localization_loss
tag: Loss/localization_loss



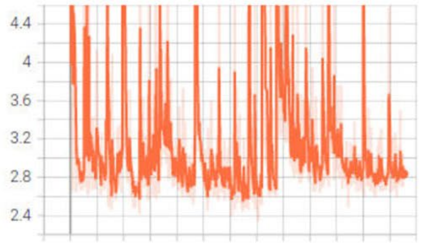
Loss/normalized_total_loss
tag: Loss/normalized_total_loss

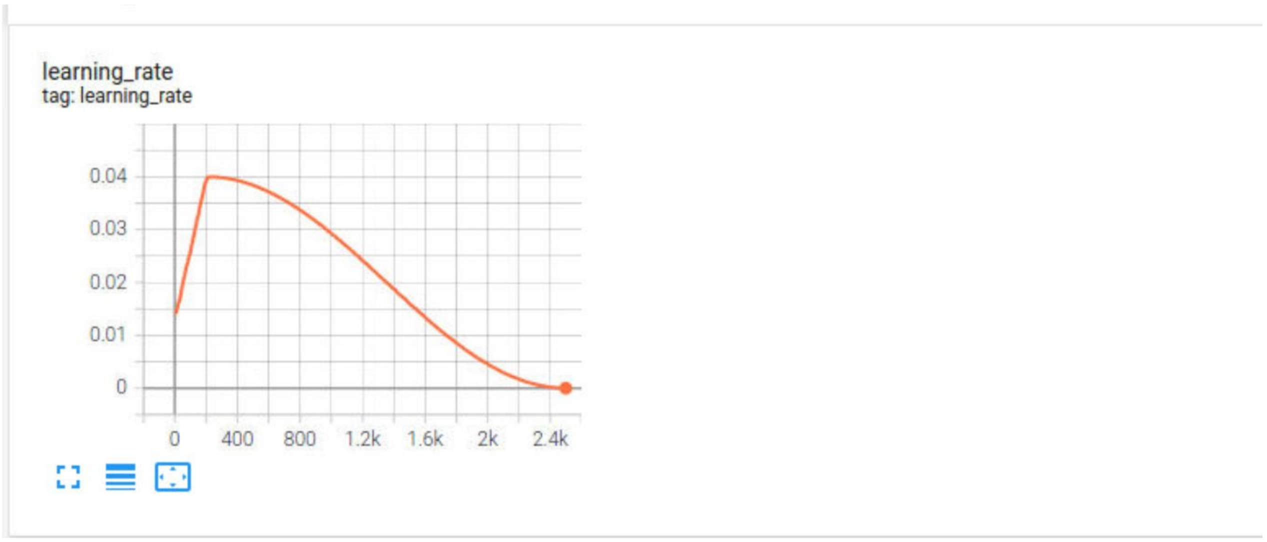


Loss/regularization_loss
tag: Loss/regularization_loss

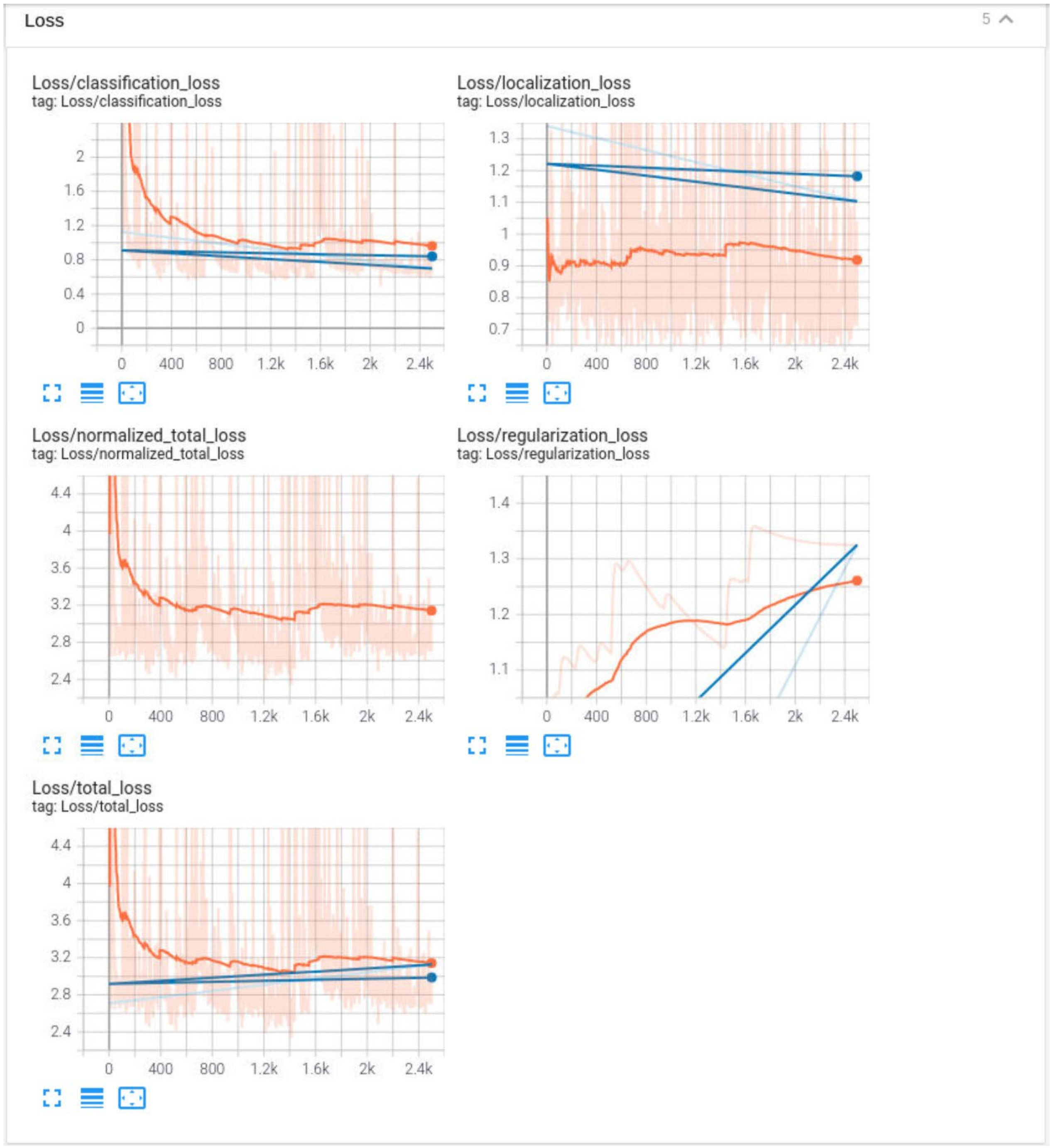


Loss/total_loss
tag: Loss/total_loss

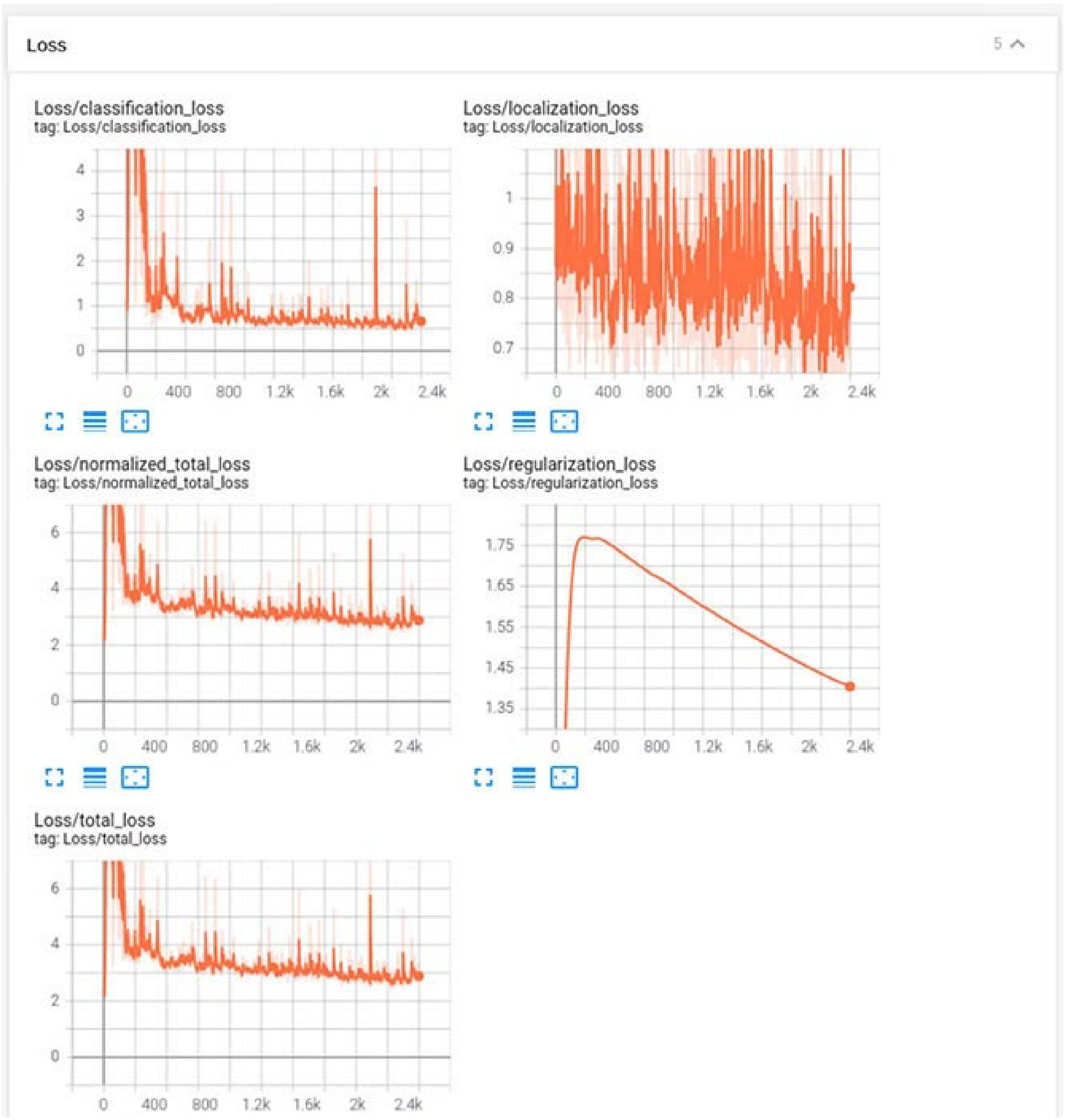




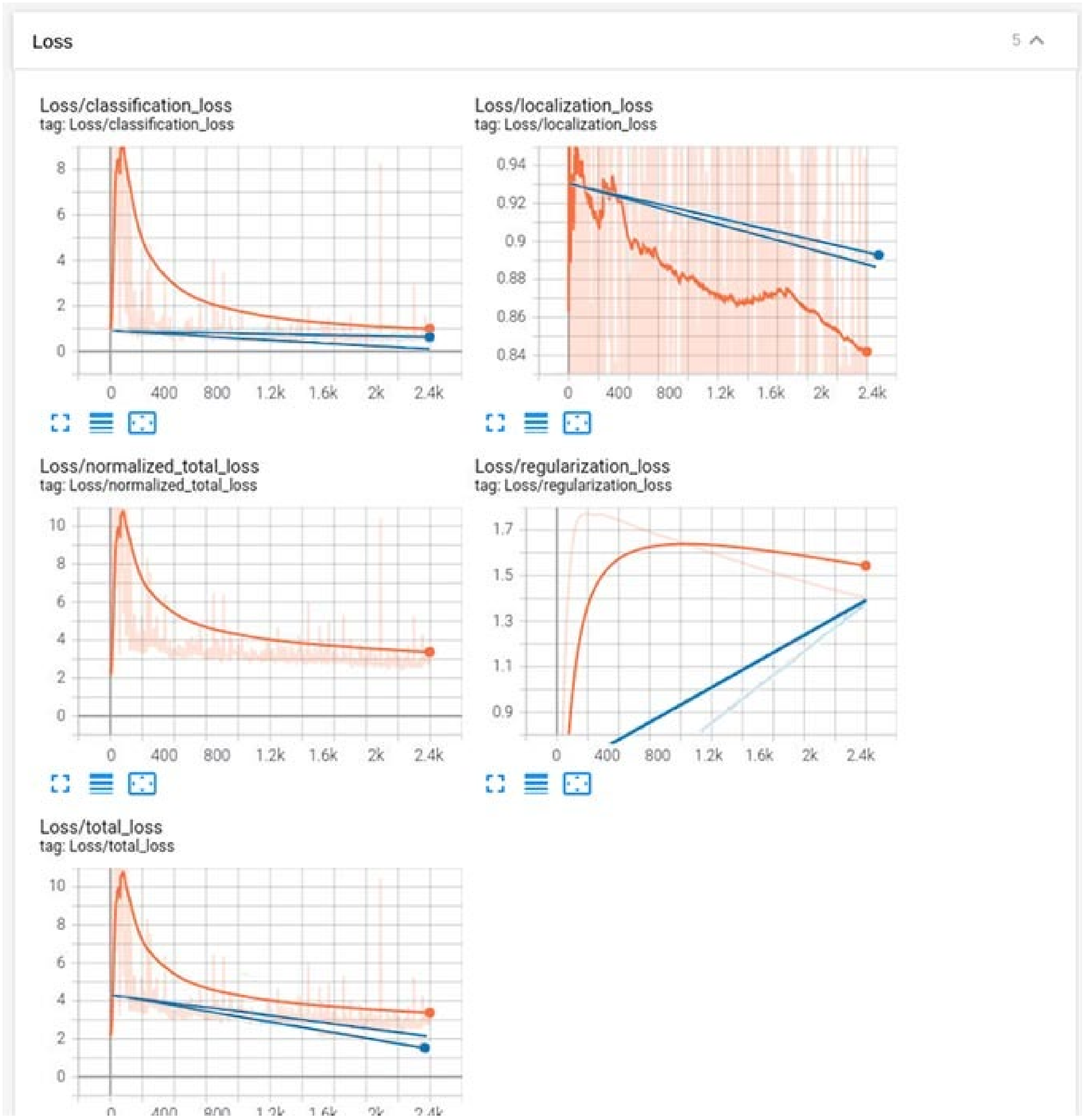
The results for the evaluation were pretty much similar to the results of the training as shown in the following figure.



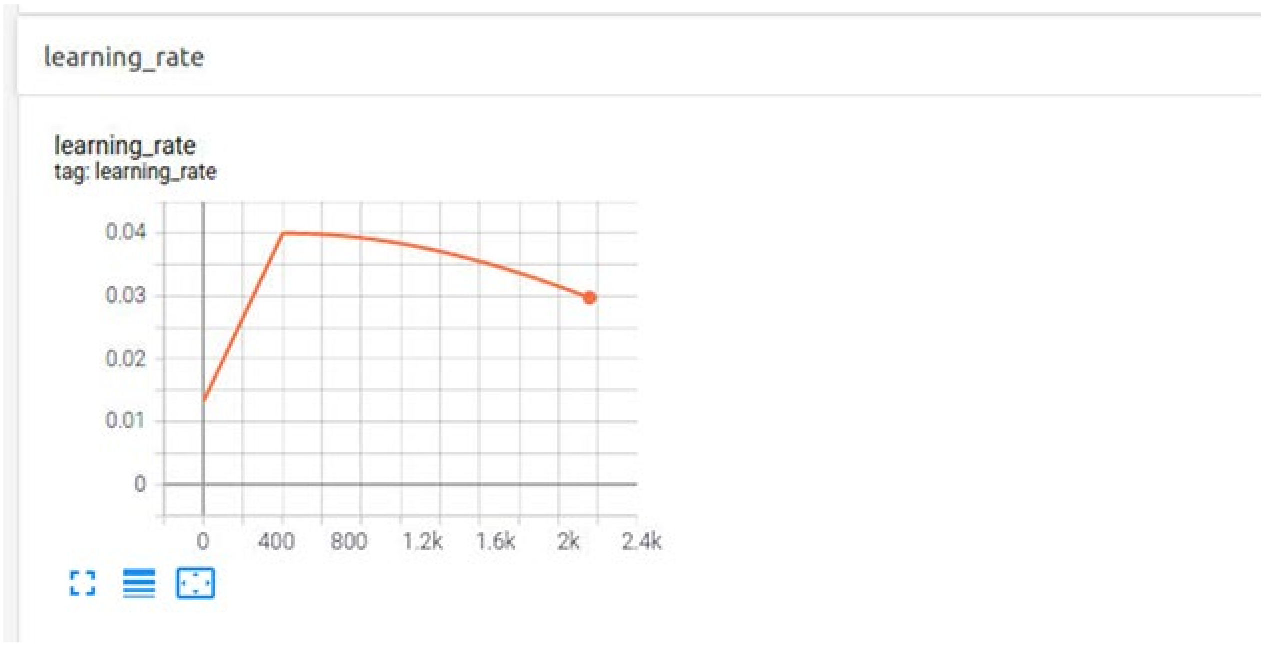
For the next experiment, I introduced the random_rgb_to_gray augmentation. This resulted in a noticeable improvement in the localization loss. However, total losses remained around 3.2 as can be seen in the graphs below.


Loss
5
Loss/classification_loss
tag: Loss/classification_loss
4
3
2
1
0
0
400
800
1.2k
1.6k
2k
2.4k
Loss/localization_loss
tag: Loss/localization_loss
1
0.9
0.8
0.7
0
400
800
1.2k
1.6k
2k
2.4k
Loss/normalized_total_loss
tag: Loss/normalized_total_loss
6
4
2
0
0
400
800
1.2k
1.6k
2k
2.4k
Loss/regularization_loss
tag: Loss/regularization_loss
1.75
1.65
1.55
1.45
1.35
0
400
800
1.2k
1.6k
2k
2.4k
Loss/total_loss
tag: Loss/total_loss
6
4
2
0
0
400
800
1.2k
1.6k
2k
2.4k

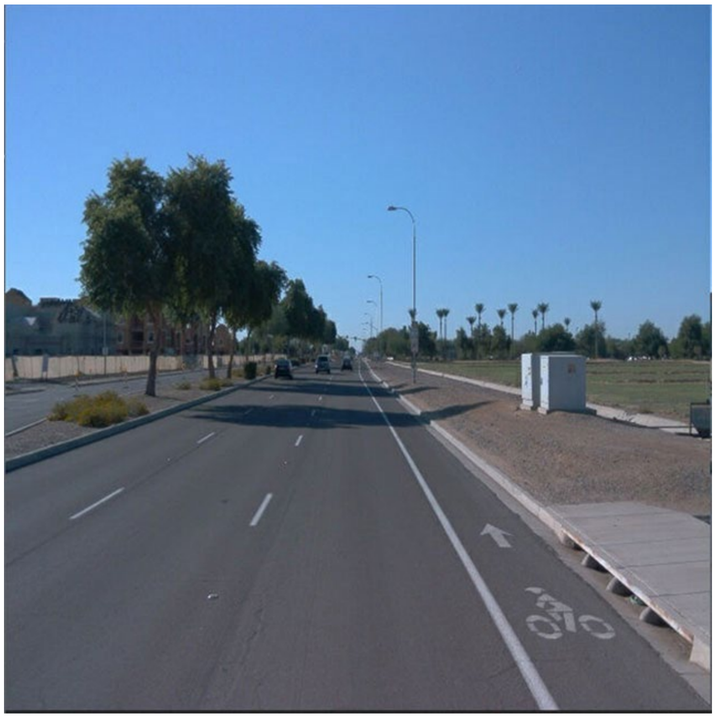
The numbers did come down slightly for the evaluation as shown in the graph below.



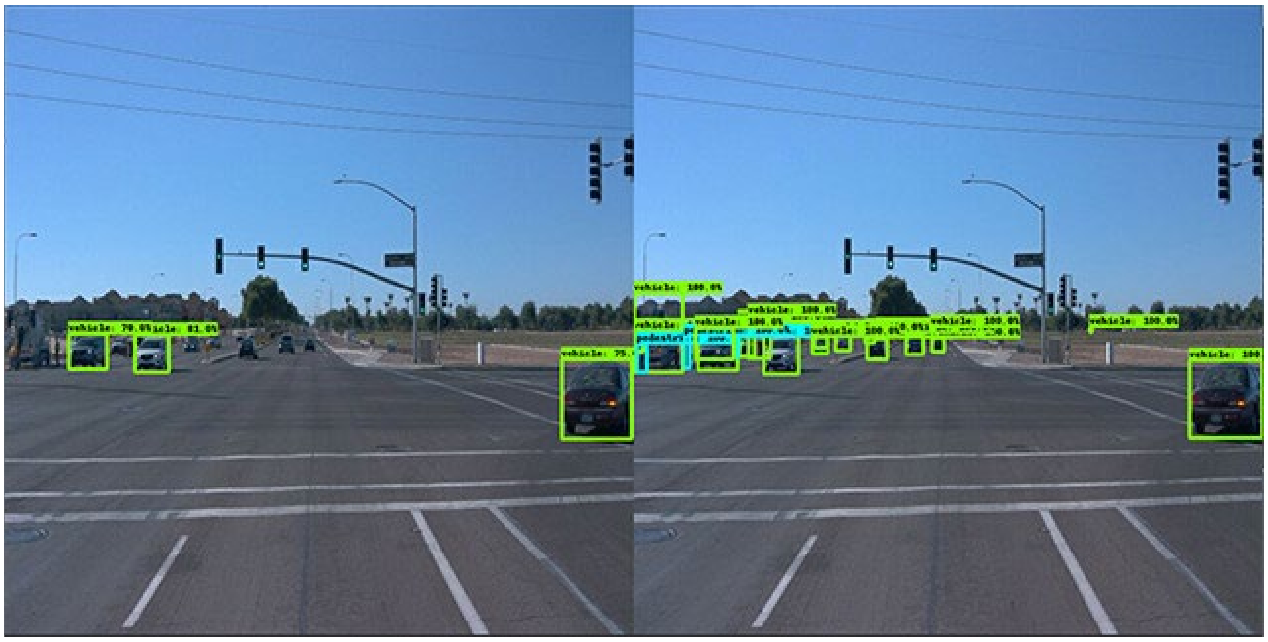
There was a vast improvement in the learning rate although it did tend to come down toward the end of the training.



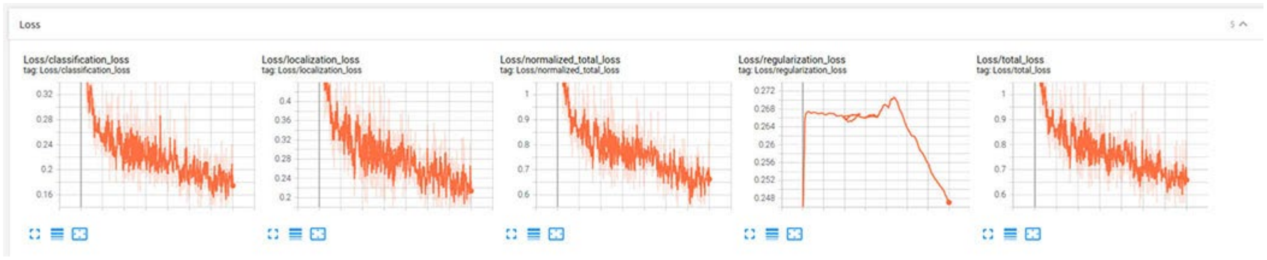
For the next experiment, I added even more augmentations. I noticed that the model had been struggling with detection where too many shadows were involved or in high brightness situations as shown in the picture below.



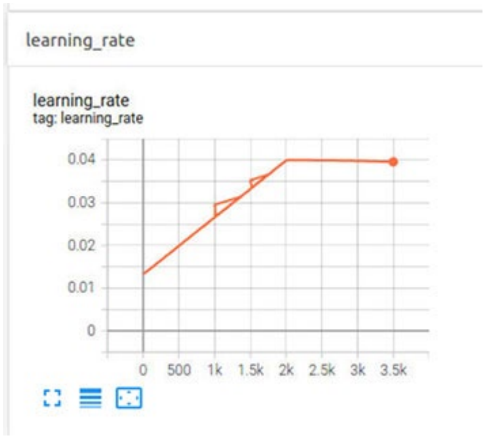
I added the random_adjust_contrast and random_adjust_brightness augmentations to the config file and trained the model again. The difference was drastic. Here is a side by side picture of same scene before and after training.



Losses were also significantly lower all across the board dipping very close to 0 as can be seen in the chart below.



The learning rate also went up and remained consistent at 0.04 without coming down


learning_rate
learning_rate
tag: learning_rate
1k
1.5k
2k
2.5k
3k
3.5k