Object Detection in Urban Conditions

Author: Joao Silva, joao.p.silva@mercedes-benz.com
Date: 2022-12-17

Project Scope

The goal of this project is to perform two tasks, being detection and classification of objects in 2D Images. The objects are from the Automotive domain, and will consist of vehicles, pedestrians and cyclists. The dataset used is coming from Waymo, and contains the images and label data specifying the ground truth (GT) metadata, for example where the objects are and what is their bounding boxes (expected result of the detection task) and what is their classification (expected result of classification task). These metadata files are recorded in .tfrecord files, and all the project will be based on the TensorFlow Object Detection API Framework.

We will initially perform Exploratory Data Analysis to see with which data are we dealing with, then we train a first reference model with a given training pipeline configuration file, and finally we make some experiments to improve the model performance by either adapting the model hyper-parameters or by augmenting our training data.

Exploratory Data Analysis

Split set

In our case the data split is already pre-done. We were given a split distributed with 86 training records, 10 validation records and 3 testing records. This is a customary split distribution, in the 90%/10% range. Other approaches can also be used, example of a 80%/20% split. Ideally, the validation/test folders are supposed to be black boxes to which we are not allowed to peak at, and we only perform evaluation on it. The first step of the exploratory data analysis is then to peak at the training records set and see with which kind of data are we peaking at. A high level summary of the training dataset it can be seen in the following Figure, where we see the 86 record files:

```
In [2]: dataset = get_dataset("/home/workspace/data/train/*.tfrecord")

INFO:tensorflow:Reading unweighted datasets: ['/home/workspace/data/train/*.tfrecord']
INFO:tensorflow:Reading record datasets for input file: ['/home/workspace/data/train/*.tfrecord']
INFO:tensorflow:Number of filenames to read: 86
WARNING:tensorflow:From /data/virtual_envs/sdc-c1-gpu-augment/lib/python3.7/site-packages/object_detection/builders/da
taset_builder.py:105: parallel_interleave (from tensorflow.python.data.experimental.ops.interleave_ops) is deprecated
and will be removed in a future version.
Instructions for updating:
Use `tf.data.Dataset.interleave(map_func, cycle_length, block_length, num_parallel_calls=tf.data.experimental.AUTOTUNE
)` instead. If sloppy execution is desired, use `tf.data.Options.experimental_deterministic`.
WARNING:tensorflow:From /data/virtual_envs/sdc-c1-gpu-augment/lib/python3.7/site-packages/object_detection/builders/da
taset_builder.py:237: DatasetV1.map_with_legacy_function (from tensorflow.python.data.ops.dataset_ops) is deprecated a
nd will be removed in a future version.
Instructions for updating:
Use `tf.data.Dataset.map()
```

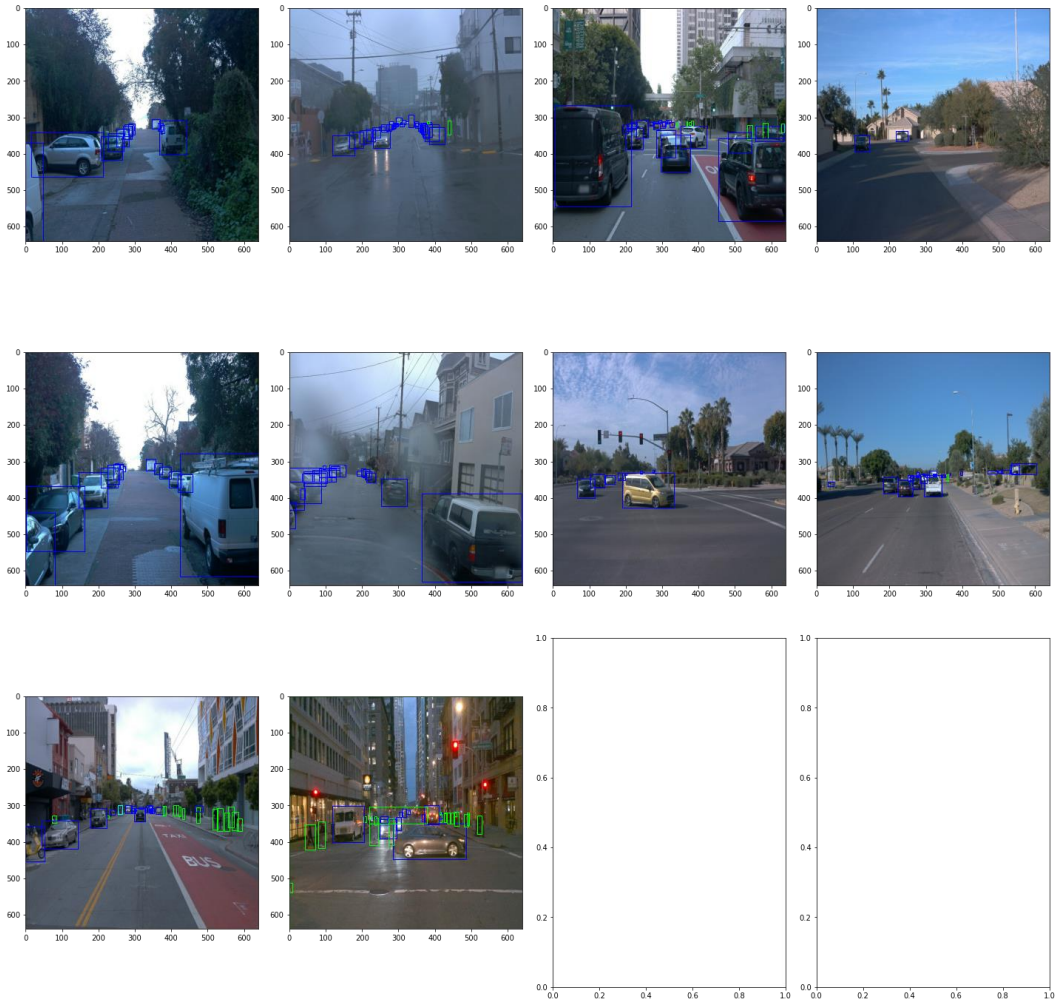
Batch attributes

We can then randomly pick (using shuffle) and grab a batch of N frames. A batch has the attributes shown in the following Figure. One interesting thing to see is that many properties are ground truth related, with example the confidence, the boxes, the classification, which can be used in a smart way to train the model.

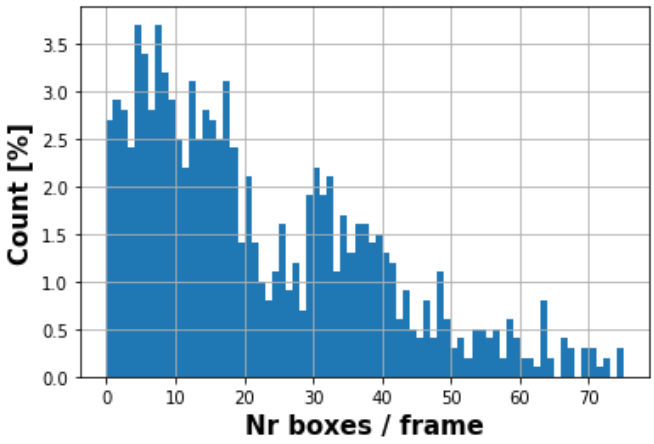
```
<DatasetV1Adapter shapes: {image: (None, None, 3), source_id: (), key: (), filename: (), groundtruth_image_confidences
: (None,), groundtruth_verified_neg_classes: (None,), groundtruth_not_exhaustive_classes: (None,), groundtruth_boxes:
(None, 4), groundtruth_area: (None,), groundtruth_is_crowd: (None,), groundtruth_difficult: (None,), groundtruth_group
_of: (None,), groundtruth_weights: (None,), groundtruth_classes: (None,), groundtruth_image_classes: (None,), original
_image_spatial_shape: (2,)}, types: {image: tf.uint8, source_id: tf.string, key: tf.string, filename: tf.string, groun
dtruth_image_confidences: tf.float32, groundtruth_verified_neg_classes: tf.int64, groundtruth_not_exhaustive_classes:
tf.int64, groundtruth_boxes: tf.float32, groundtruth_area: tf.float32, groundtruth_is_crowd: tf.bool, groundtruth_diff
icult: tf.int64, groundtruth_group_of: tf.bool, groundtruth_weights: tf.float32, groundtruth_classes: tf.int64, ground
truth_image_classes: tf.int64, original_image_spatial_shape: tf.int32}>
```

Batch analysis

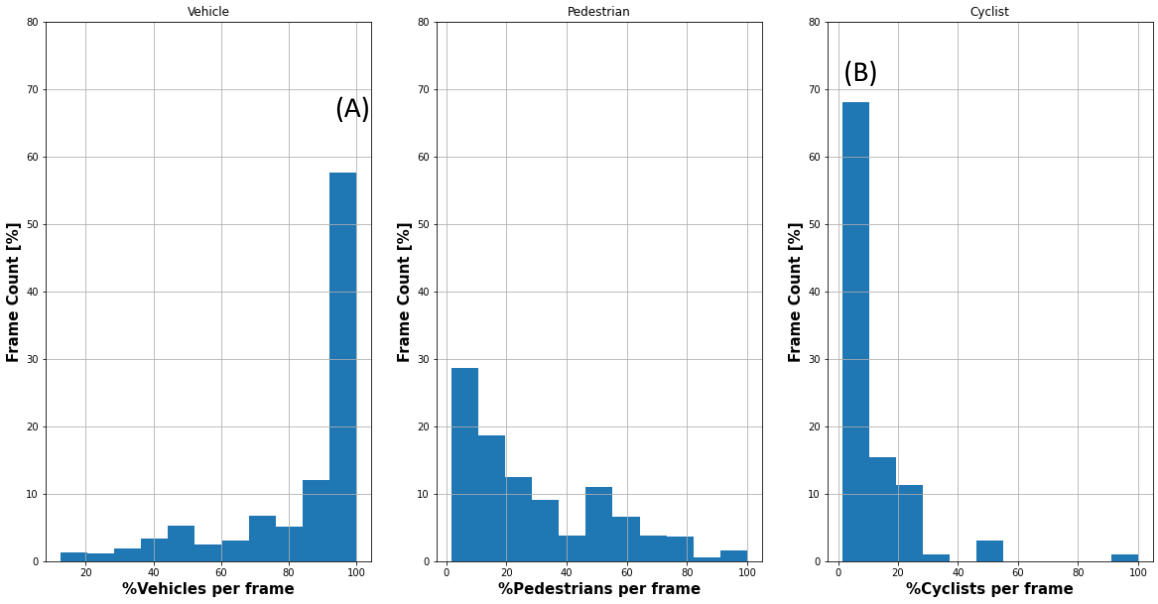
An example of a random batch with 10 frames can be seen in the following Figure. We can see the multitude of bounding boxes as well as the classification of them, color coded. We can see variance in the frames, such as different objects (cars and pedestrians), different density of objects (crowded and sparse) or weather conditions (foggy vs clear, daylight vs nighttime). This indicate that the database will be hopefully proper to train with. Nevertheless, we see no cyclists, which indicates some problems might arise for its classification.



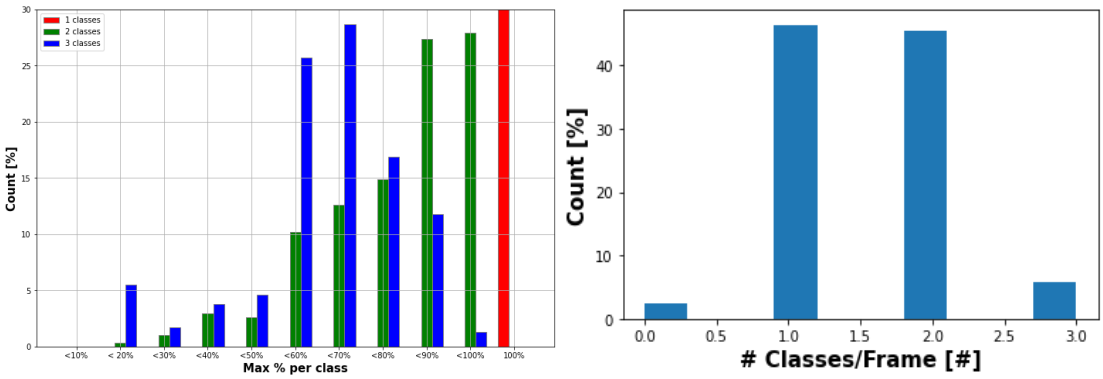
We make a further analysis to try to validate our previous statements. Firstly, we check what is the density of the label boxes. We conclude we have a good mix, ranging from very sparse (<10 boxes/frame) to very dense (>50 boxes/frame). We see that in limit, around a maximum of 75 boxes/frame can exist, and this can be used to help with our training speed. We can see as well a good amount of frames ranging in the middle, and the distribution correlation is approximately linearly inverse.



Secondly, we check for the density of labels. We can validate our previous statement by checking that there are much more density of vehicles than pedestrians than cyclists. For instance, about 60% of the frames represent frames with vehicles consisting of >90% of the bounding boxes (A). In addition, about 75% of the frames represent frames with cyclists consisting of <10% of the bounding boxes (B).



Finally, we check the classification mixture, that is, we evaluate for each frame what is the maximum percentage of each class per total classes available in that frame. For only 1 unique class frames, the maximum percentage would be obviously 100%. For 2 or 3 unique class frames, we could assume a good mixture if at most a class would have a maximum percentage of 50%. For all this cases, we catch very less frames, nearly below 5% of the respective. We also analyze the #Classes/Frame, and we can see that some very less frames have either 0 classes or 3 classes, but for 1 or 2 classes seems to be quite uniform.



Model Training

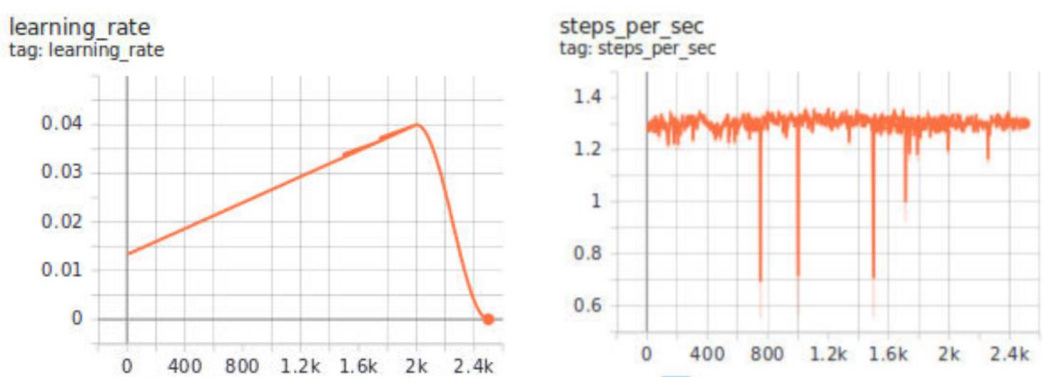
Training and Evaluation steps

We are very limited because of the workspace storage space and GPU time. Therefore, we adapted both the training steps number in the pipeline configuration file and the steps per checkpoint in the main tensor flow model file for each individual training/evaluation process (not only for the reference model but as well for the improvements). The first parameter is due to the GPU time limitation, while the other is to add more granularity to the evaluation step which impacts the storage space. This generates more intermediate data, and to deal with that, we copied the intermediate checkpoint files to /home/, although it is not maintained between sessions.

Reference Model

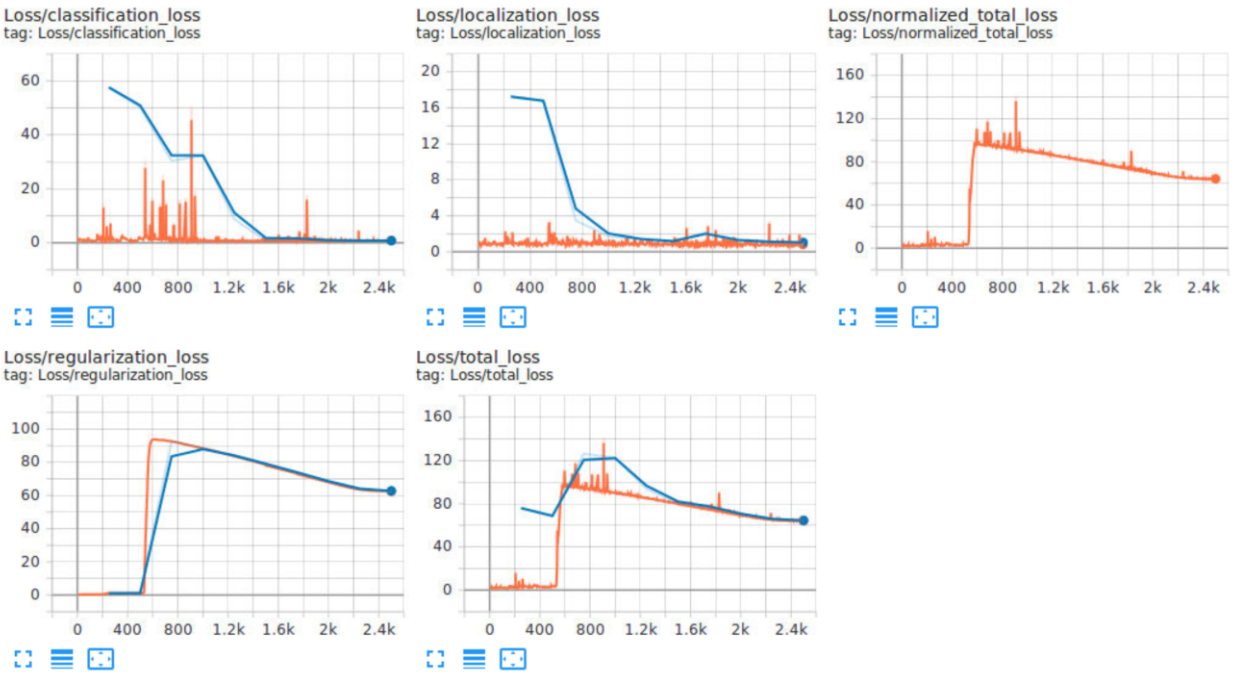
The first training session is called the reference model. This will be the benchmark to which we will compare our results to. The pipeline configuration file is given, so at initial step we just need to let the model train, monitor it, and evaluate it. The monitoring and analysis of results is used with the help of the Tensor Flow dashboard, called TensorBoard. The results can be seen in the Following Figures.

Firstly, we see the model learning rate curve, which is as specified in our configuration file. Starts at 0.013, then ramps up in 2000 steps until reaches 0.04, and then drops to 0 using a co-sine decay rate. We also observe that for our configuration file, each training step takes about 1.3 seconds.

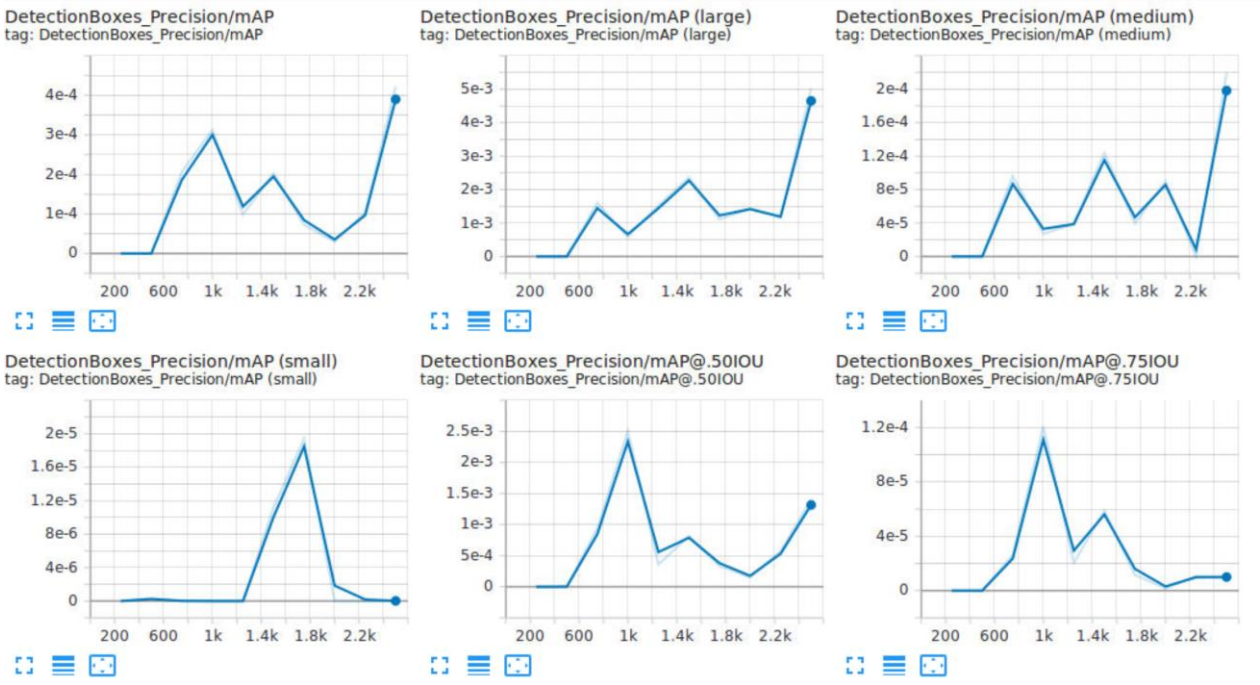


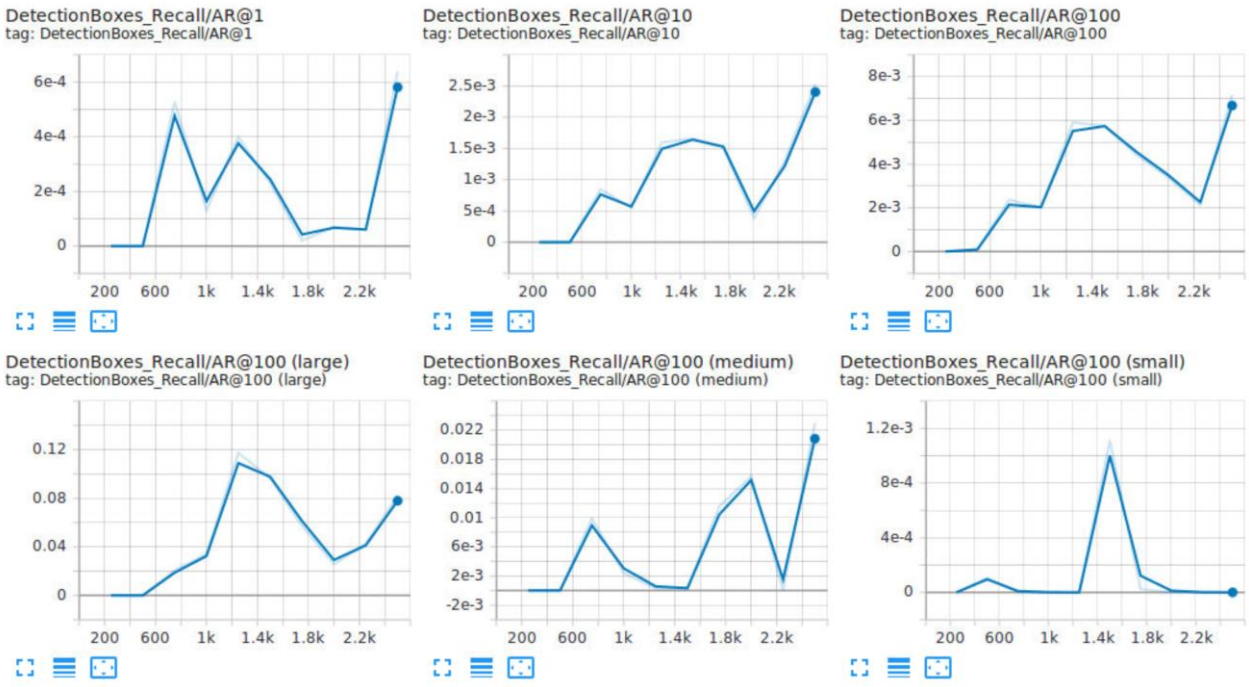
Secondly, we see what is the model loss step by step and compare it to the validation loss. We can see that initially the evaluation loss, for classification, localization and regularization (blue) is much higher

than the training loss (orange), which might indicate overfitting, as the training performes better than the validation. After some steps, we see a convergence between both.



Finally, we evaluate the two major indications, precision and recall. We see in all values very small, well below even 1%. The only exception is the Recall of very large objects, sitting around 8%, which still is not great. Also, we don't see an incremental increase in performance. All these condition indicate that this reference model is poor. Nevertheless, we can still use it for baselining, and we expect in the next chapter to achieve better results.





Model Improvements

Model Hyper-parameter improvements

Now that we have a baseline, we can try to improve our model and compare it to it. Our first try is to change some model hyper parameters.

Results of the 1st improvement

Batch Size

The first improvement we tried was the batch size. We saw that on the original configuration file the batch size was set to 64. On the reference file it was set to 2. The reason is to decrease compuntational resources (both time and GPU). So we tried to at least increase it to improve performance. At first we tried with 30 and 20 but the training process crashed. Finally we tried with 10 and it is feasible.

Learning parameters

At the same time, we increase the learning start-up rate of the optimizer used. The idea is to try to make the learning process faster, and then stabilize smoother.

Discussion

Using both the above mentioned Batch Size and Learning Parameters changes, we let the traing re-run. The results can be seen in the following Figures.

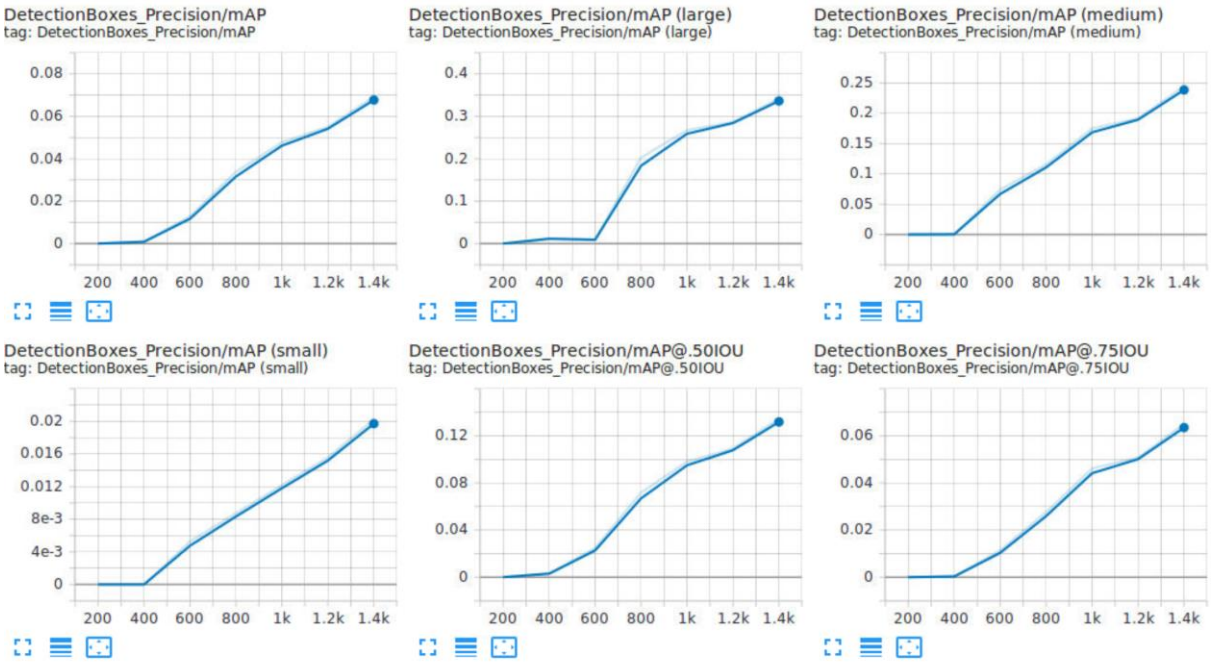
We can see that the results improved. We now have some precision and recall metrics in the range of 10%-40%. This is not yet sufficient, usually a value above 50% is needed, but definitely there is an improvement with respect to the baseline.

We can also see that the precision/recall curves are still increasing. This indicates that we might have stopped training too early, as it seems the metrics are still being improved with time. This can also be seen

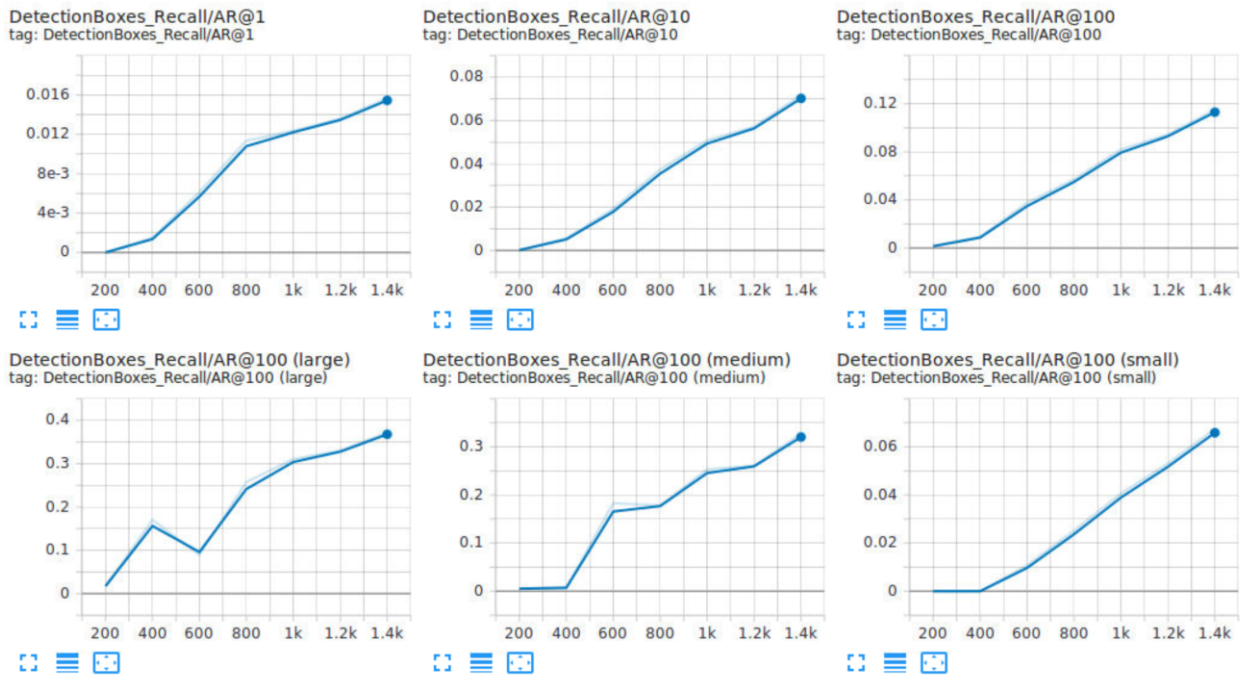
in the Loss curves, that definitely improved and now are steadly decreasing, but it doesn't look that converged was already achieved.

Finally, we can see that the step/s decreased from 1.3 to 0.4, a x3 decrease. This has much to do with the batch size, which was increased x5.

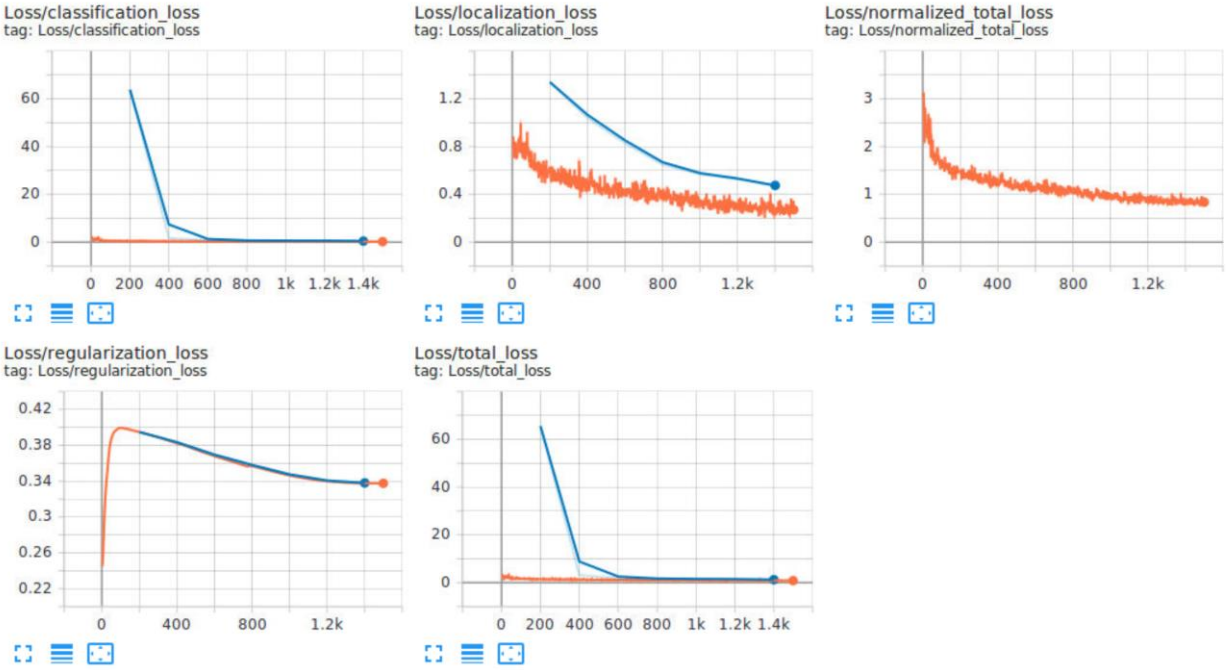
Precision Figures for 1st improvement:



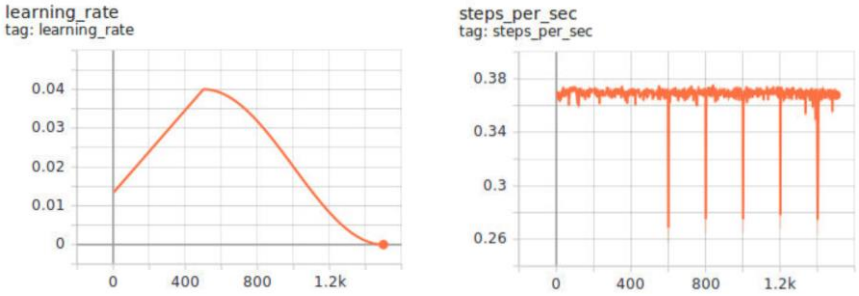
Recall Figures for 1st improvement:



Loss Figures for 1st improvement:



Learning Rate and Steps/s Figures for 1st improvement



Results of the 2nd improvement

Batch Size

This time we again increase the batch size, 10 to 15, in order to see if the performance increases further.

Learning parameters

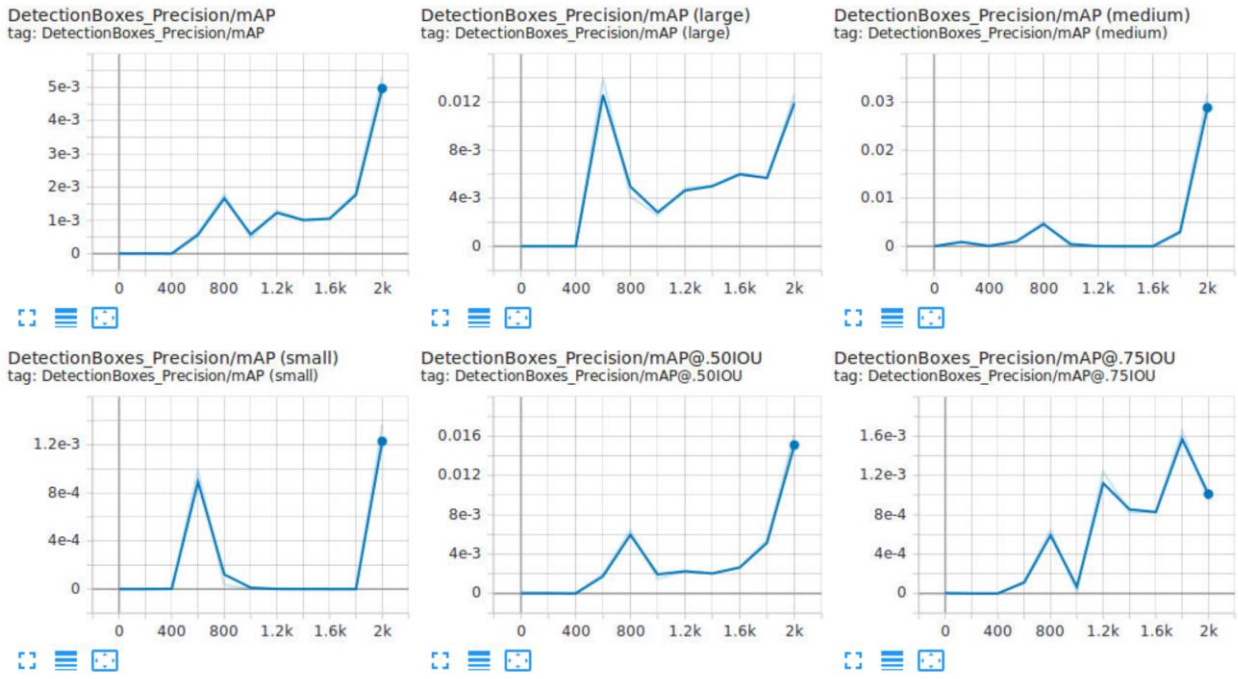
As we saw before, we had not achieved convergence. Therefore, we increased the learning steps, from 1500 to 2000, and the learning parameters, namely the warmup rate and the base rate. Nevertheless, to not be so drastic, we allow 1000 steps until base learning rate, since anyhow we have more time to learn, it is better not to add such relative increases.

Discussion

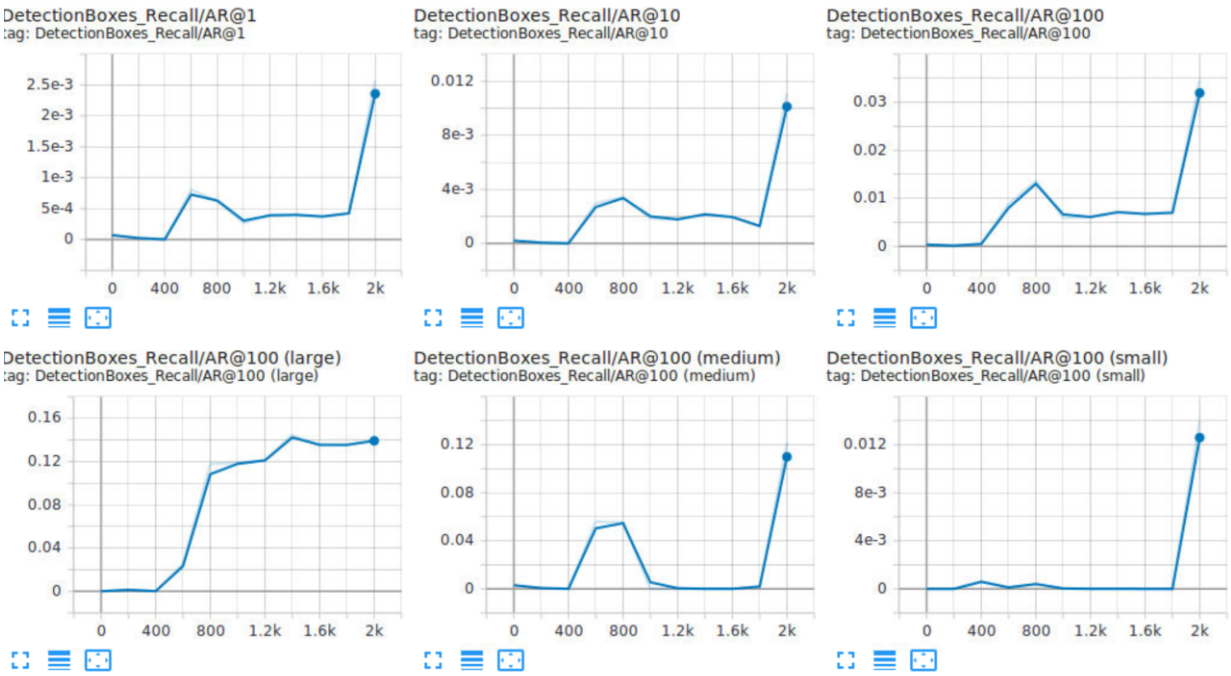
This time the results were not as satisfactory and we obtained results similar more similar to the baseline. We believe the reason lays in an too aggressive starting learning rate, that might have converged to a local minimum. This is indicated by the Total loss heading towards a plateau. To prove it, we would still let the model run some more steps.

Again, we can see that the step/s further decreased, as a increase in batch size occurred.

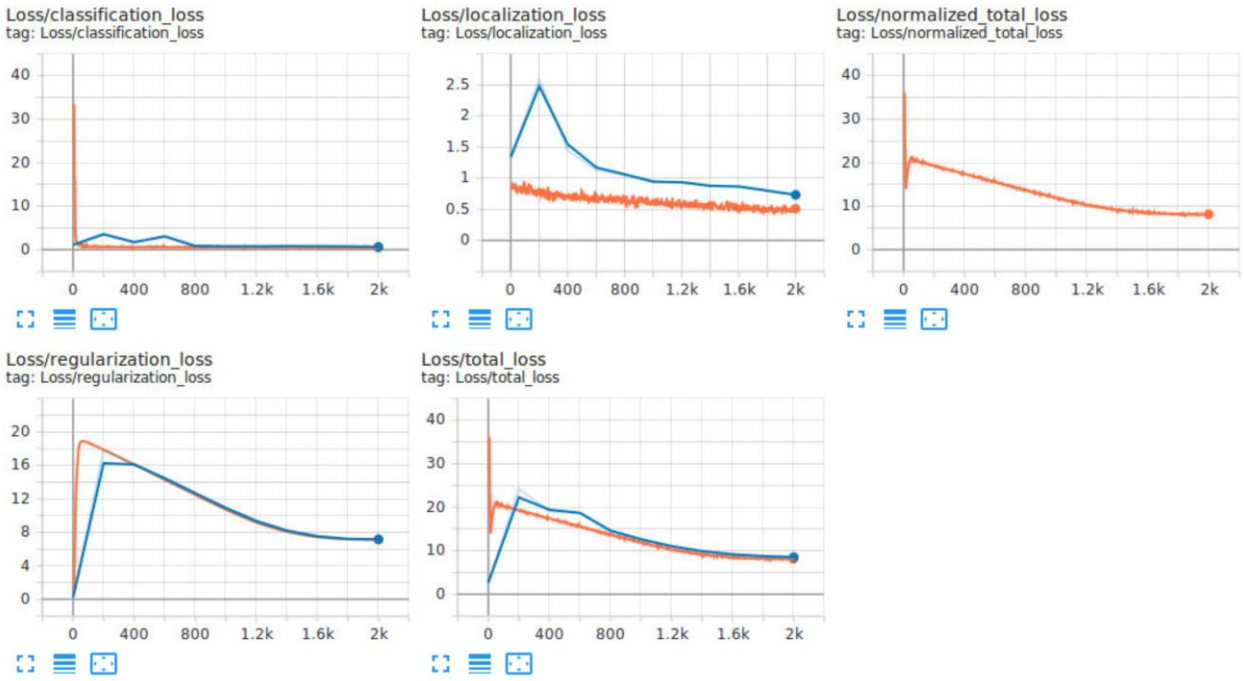
Precision Figures for 2nd improvement



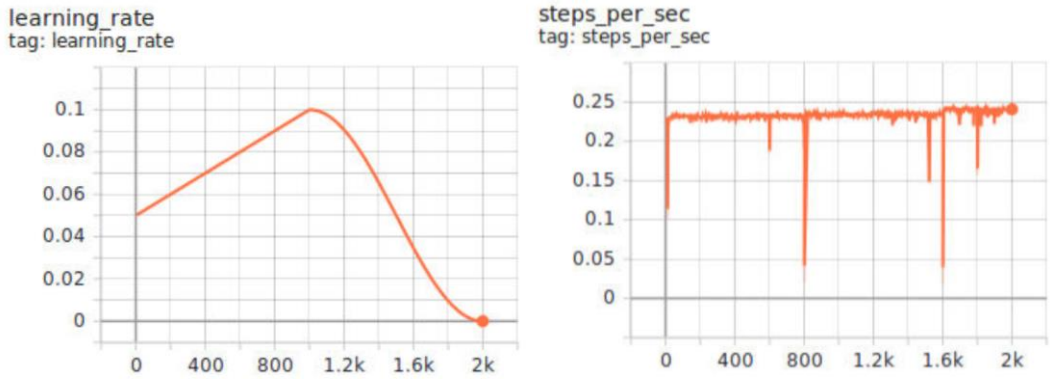
Recall Figures for 2nd improvement



Loss Figures for 2nd improvement



Learning Rate and Steps/ Figures for 2nd improvement



Data Augmentation

Data augmentation Strategies

The final step in the project is to try to augment the training data in order to achieve better results. We can increase the variety of the data, in order to give more use-cases and ways of the Model to understand the different phenomena such weather conditions. Therefore, we tried some augmentation strategies. Randomized horizontal flip and image crop were already in the predefined configuration. So we added a few more, such has:

- Random RGB to gray (simulates nigh conditions)
- Random contrast adjustment (impact on color/texture)
- Random brightness adjustment (simulates possible direct sunlight)
- Random saturation adjustment (impact of color/texture)
- Random hue adjustment (generic visibility decrease)
- Random black patches (simulates obstructions)

Some results of this augmentations can be seen in following Figures:

Batch Size

Since no great achievement was seen with an increase of the batch size from 15 to 10, but the step time increased, we reduced the batch size back to 10, the one of 1st improvement.

Learning parameters

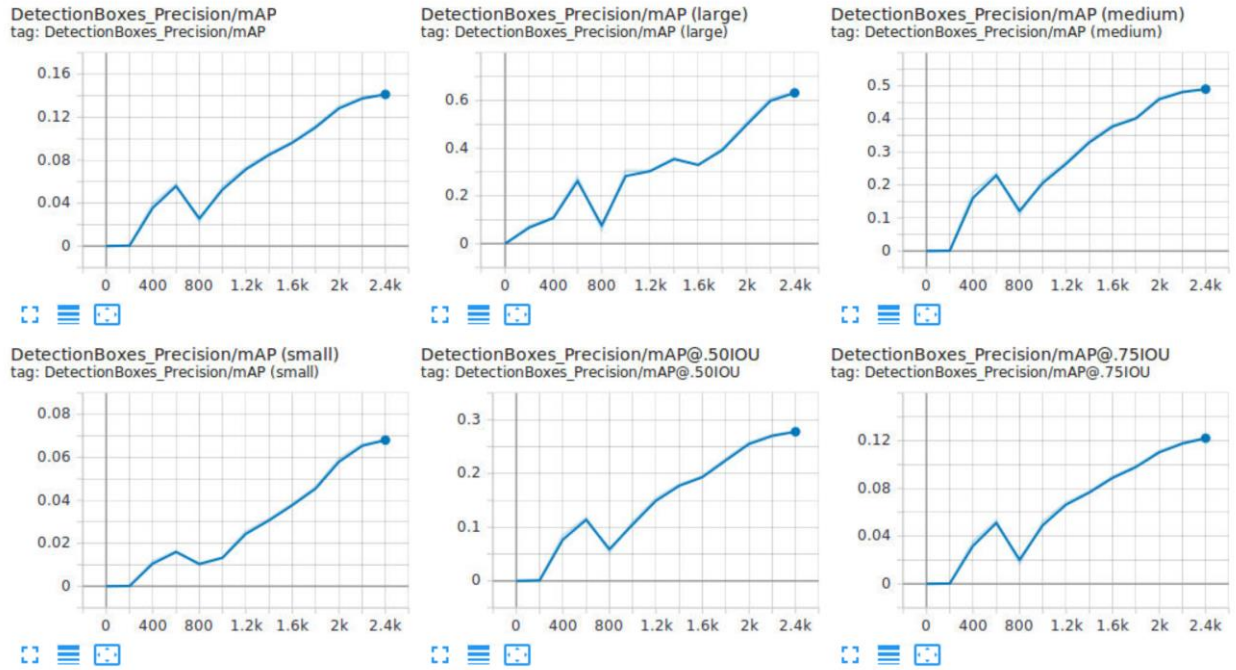
For the learning parameters we also changed back to the ones of the 1st improvement. The idea is to make them on the same base, to try to make a direct comparison with respect to the augmentation improvements. The only difference here is that we allow a bit more time to reach the base learning rate, and then we let the model train for a longer time (we had seen before that we had not reach convergence).

Discussion

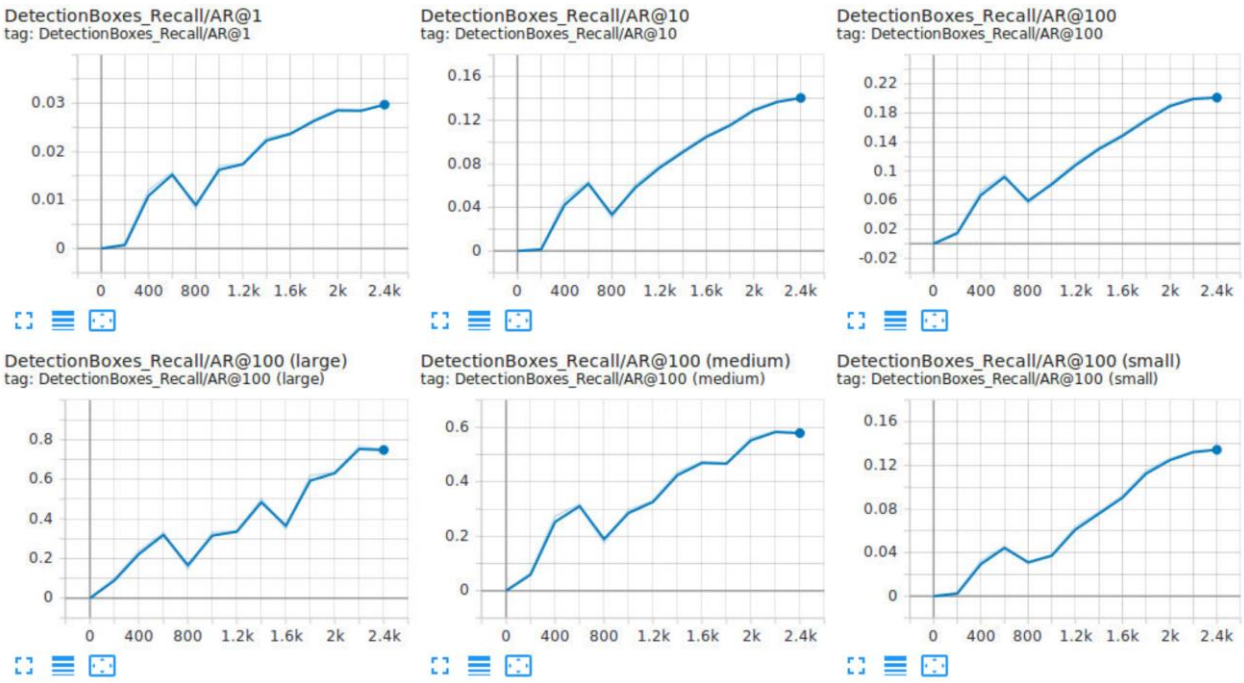
This time the improvement can be seen, and the result is the best model of the project. We can see some precision metrics in the range 50%-60% for medium and large objects, and recalls in the order of 60%-80%, again for medium and large objects. For small objects the metrics are not so good, but still perform better than the previous models.

The metrics also indicate that we have not yet reached convergence, and training the model for longer time might have gotten better results. We could not train it more as our GPU time ran out.

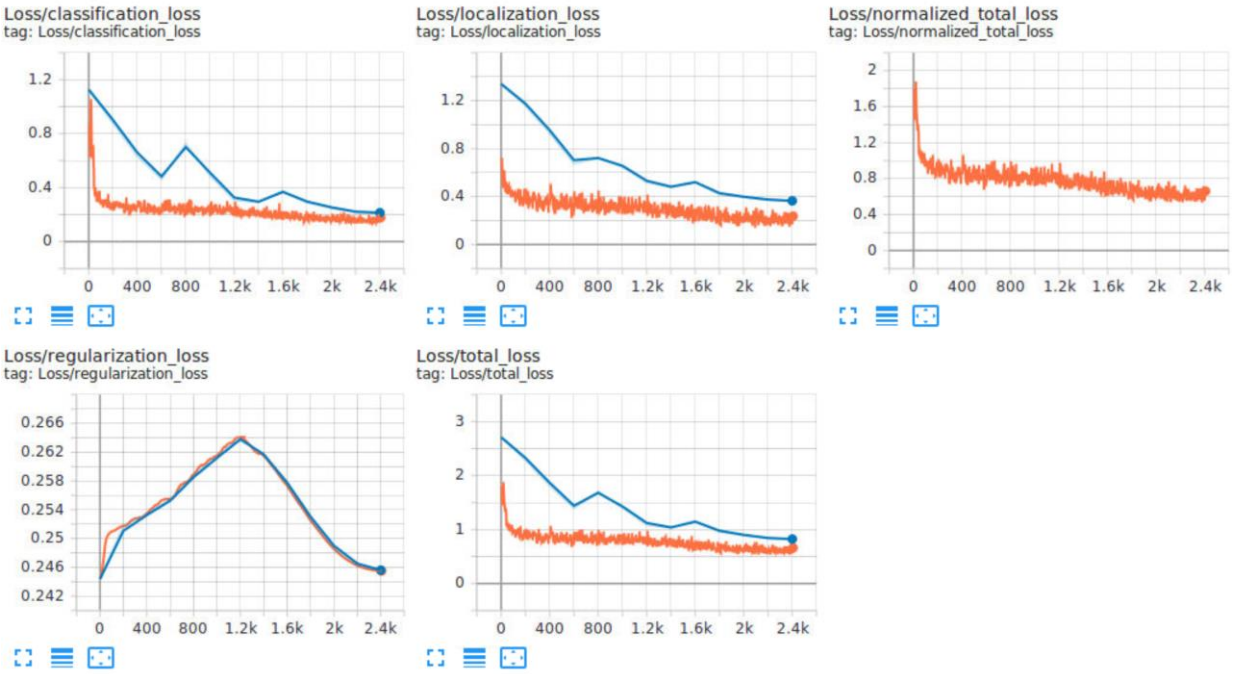
Precision Figures for Data Augmentation Improvements:



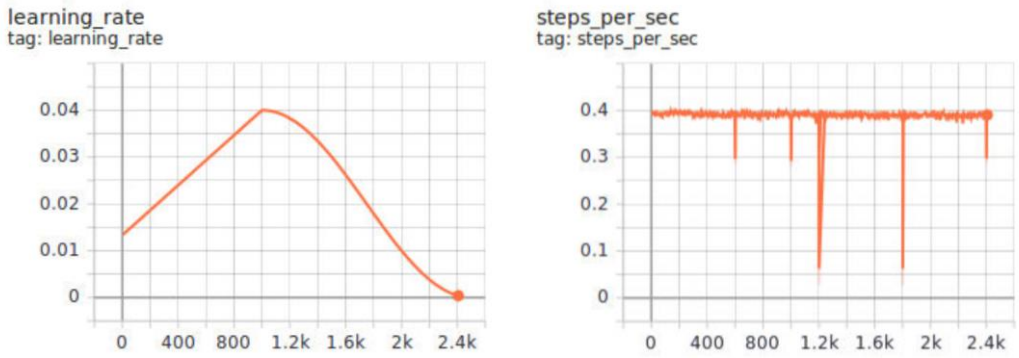
Recall Figures for Data Augmentation Improvements:



Loss Figures for Data Augmentation Improvements



Learning Rate and Steps/s Figures for Data Augmentation Improvements



Conclusion

Finally, we can conclude by stating that the objectives of the project were achieved, and we successfully trained a model to perform object detection and classification tasks. These results can be seen in the above mentioned Figures and the attached Video (see some spans in the next page). We could have achieved better results by improving more the parameters, such as learning rate but also layer size or optimizer, for example.


vehicle: 40.0%
vehicle: 66.0%
vehicle: 93.0%
vehicle: 79.0%
vehicle: 31.0%
vehicle: 52.0%
vehicle: 65.0%
vehicle: 57.0%
vehicle: 62.0%