Object Detection Project Writeup

Submission by: **Nolan Lunscher**
Submission Data: **2021-09-21**
Repo: **https://github.com/nlunscher/nd013-c1-vision-starter**
Repo branch: **devel-nl**

1 Project Overview

In this project, the goal is to train a Deep Learning Object Detection Algorithm to perform bounding box object detection in Urban Environments. To accomplish this task we are utilizing the Tensorflow Object Detection API. In order to speed up and stabilize training, we utilize Transfer learning from a model previously trained on the COCO dataset.

Through this project we are looking to explore a real world model training pipeline, as well as gain experience with how to improve model performance.

Object detection is an important component of self-driving car systems in that it provides useful information describing a vehicle's surroundings. Specifically, object detection provides a vehicle with information about what objects are around it, as well as roughly where those objects are. This information is necessary for the vehicle to be able to make decisions about how to safely navigate an environment.

Bounding Box Object Detection on its own is confined to 2D detection in images, which does not fully describe an object's position. This can however be combined or expanded with other sensor inputs and algorithms to produce true 3D object detection, which would allow for a vehicle to use this information in its navigation system.

Project Setup

After cloning the git repo and checking out the correct branch, build the docker image using the **Dockerfile.gpu** in the **build** folder. The tfrecord data files should be placed in "**../data/waymo_processed**" relative to the repo folder.

2 Dataset

2.1 Dataset Analysis

More details can be found about this section in the **Exploratory Data Analysis.ipynb** file in my repository.

Figure 2.1-1: Sample of images and label inspection from the dataset. Red: Vehicle, Green: Pedestrian, Blue: Cyclist.

By visual inspection of the data, it can be seen that there is a wide variety of detections across the images. This includes large detections that take up large amounts of the images, as well as smaller detections that are quite difficult to make out at all. It appears that the vast majority of the detections are some sort of vehicle, with pedestrians and cyclists only appearing relatively rarely. Much of the data appears to be from clear days where visibility is good, however I did run across a number of samples from dark conditions or bad weather conditions where visibility was significantly worse. In these worse conditions, it was definitely more difficult to make out the objects in the environment.

Class distribution is an important part of machine learning, as the model has the potential to focus on classes that appear more often. To get a better idea about the class distribution, I took a random sample of 10000 images, and plotted a count of the classes that appeared (shown below). From this, my initial observation was confirmed, where vehicles are by far the most common class, followed by pedestrians, and finally cyclists. This tells me that a model trained on this data will most likely perform best at detecting vehicles, and potentially struggle with the remaining two classes. This is because it will gain the most exposure to vehicle classes during training, as well as be optimized largely towards vehicle performance.

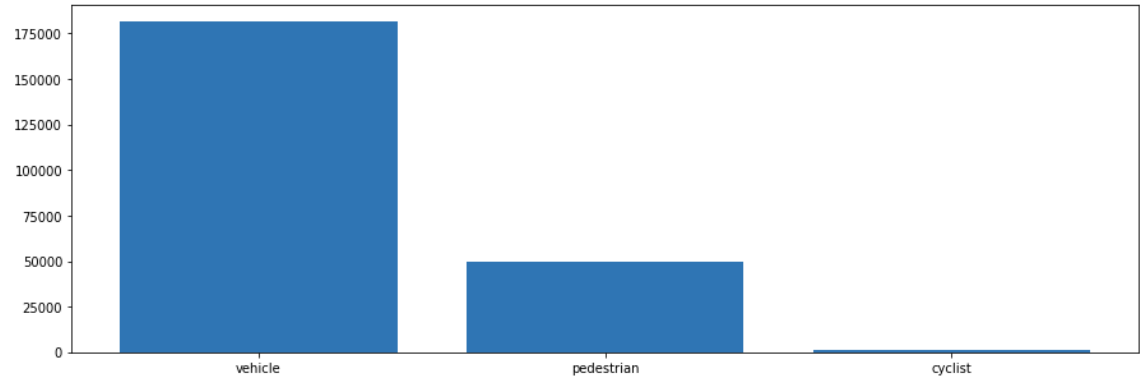

Figure 2.1-2: Class bounding box counts for 10000 images in the dataset.

Another important aspect of object detection data, is the distribution of detections by their size in images. This roughly approximates their distance to the camera. Another important piece of information about the detections is the bounding box aspect ratio. The aspect ratio of ground truth boxes is important when tuning the anchor box configurations of an object detection algorithm. This is because the anchor box shapes should resemble the boxes found in the training data, such that they can align well and have higher IoUs.

To get a better idea about the box size and shape distribution, I took a random sample of 10000 images, and used a scatter plot of the box area and aspect ratios (shown below). From the figure, it can be seen that the majority of the boxes are less than 200,000 square pixels (~448x448, if aspect ratio = 1:1). For context, I formatted the dataset image to a resolution of 1920x1080. This tells us that most detections take up a relatively small part of the image. This makes sense because it's not common for many objects to be so close to the camera as to take up the majority of the FOV.

Another observation from this plot is that the majority of the detections are relatively square, with aspect ratios <1:2.5. This is consistent with what was seen based on visual inspection,

where most detections come from vehicles, which themselves do not have any view point where their length vs height would produce a very long or tall box. I suspect that isolating the case of pedestrians, this distribution would skew more towards more rectangular boxes.

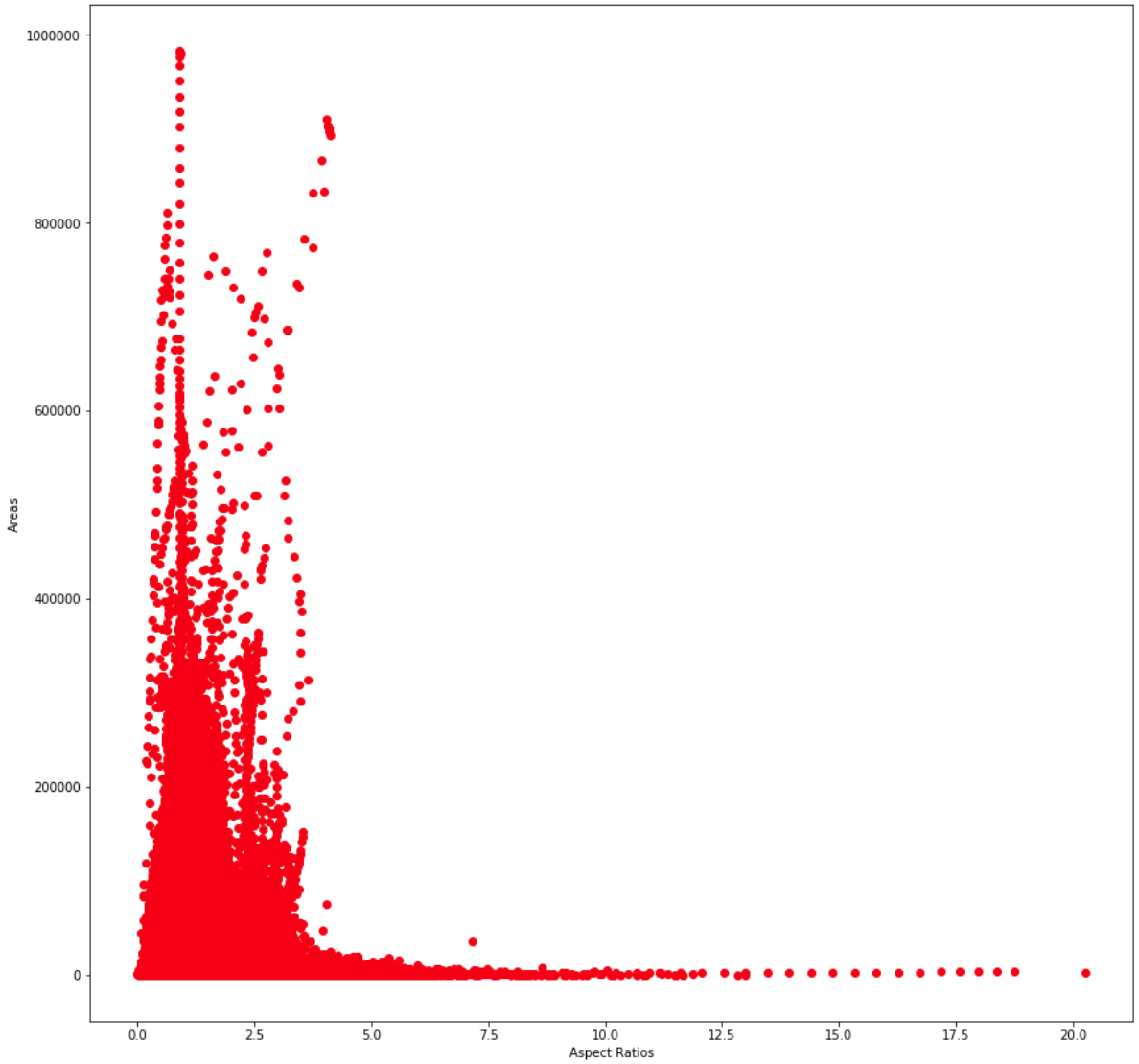

Figure 2.1-3: Bounding box area and aspect radios in 10000 images from the dataset.

2.2 Cross Validation

For the dataset splits, I chose to use a 70% training, 10% validation and 20% testing split. The majority of data is required for training, however there is a significant amount of data in this dataset, so we do not need to be too stingy on validation and testing in order to leave enough training data for convergence. Generally the test data should be large enough to provide a reasonable approximation of algorithm performance once deployed (the real testset is the real world, but that is hard to experiment with as it is not controlled). For this

reason I decided to make the test dataset larger than the validation dataset. Beyond this strategy, I selected 70-10-20 since they are nice round numbers.

My implementation can be seen in **create_splits.py**.

3 Training

3.1 Reference Experiment

My reference experiment was to train with a batch size of 8 for 25000 iterations using the remaining default configurations. Training loss viewed in tensorboard during training can be seen below. The loss consistently drops throughout the course of training, indicating that at least for the iterations performed, optimization did not get stuck in a local minima. Validation loss follows a pretty consistent downward trend throughout training indicating that overfitting is not likely. The validation loss also does not appear to have flattened out, indicating that longer training may further improve performance.

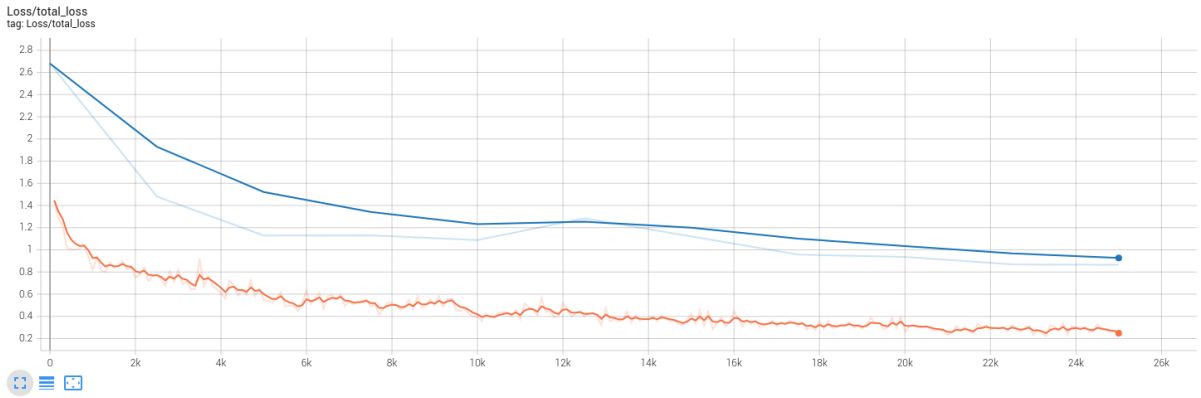

Figure 3.1-1: Training and validation loss during training.

This model training config can be found in “**training/reference**” in my github repo.

Evaluation metrics for the reference model are shown below. As can be seen they increase throughout training indicating that overfitting has not likely occurred. Similar to the trend noted in the validation loss, the validation metrics also do not appear to have flattened out, indicating that more training iterations may be able to improve performance.

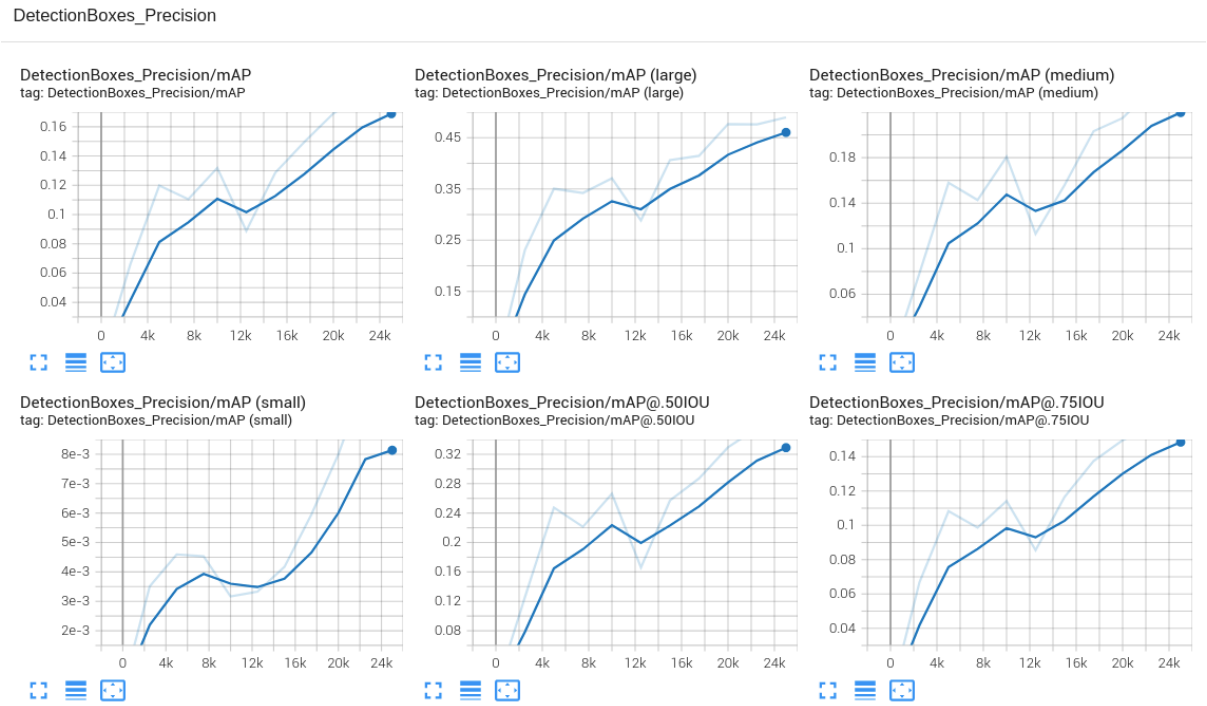

Figure 3.1-2: Reference model validation precision metrics during training.

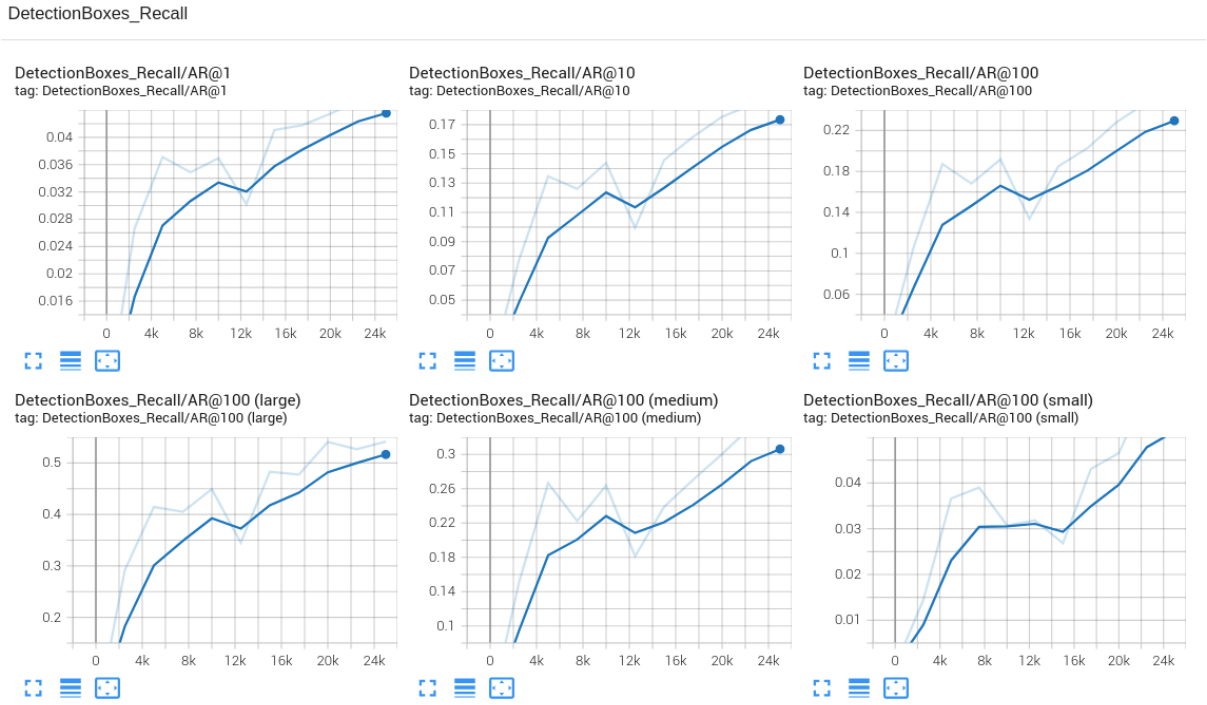

Figure 3.1-3: Reference model validation recall metrics during training.

The final reference results are shown below. Overall performance was decent overall, but there is a lot of room for improvement. It can be seen that for smaller object boxes, mAP and AR are worse than for larger object boxes. This makes sense as smaller object boxes are less distinct in images, and therefore would be harder to detect for an algorithm. As was noticed earlier in the Data Analysis section however, there are many samples of smaller object boxes in the dataset compared with larger boxes. Due to this the average results will be weighted more towards the smaller box results.

Despite low mAP and AR, the relatively higher scores for larger object boxes may mean that this algorithm is ok for deployment on a vehicle for certain scenarios. This is because generally the important objects are those that are closer and therefore appear larger in images. This would mean that for some scenarios such as short term planning, or driving at low speeds, good detections of larger object boxes are significantly more important than smaller ones.

The final evaluation on the Validation Dataset results are listed below:

DetectionBoxes_Precision/mAP: 0.168813
DetectionBoxes_Precision/mAP@.50IOU: 0.321590
DetectionBoxes_Precision/mAP@.75IOU: 0.150878
DetectionBoxes_Precision/mAP (small): 0.005956
DetectionBoxes_Precision/mAP (medium): 0.237711
DetectionBoxes_Precision/mAP (large): 0.438444
DetectionBoxes_Recall/AR@1: 0.043134
DetectionBoxes_Recall/AR@10: 0.177393
DetectionBoxes_Recall/AR@100: 0.232120
DetectionBoxes_Recall/AR@100 (small): 0.048146
DetectionBoxes_Recall/AR@100 (medium): 0.318781
DetectionBoxes_Recall/AR@100 (large): 0.507720

The final evaluation on Testing Dataset results are listed below:

DetectionBoxes_Precision/mAP: 0.173868
DetectionBoxes_Precision/mAP@.50IOU: 0.340112
DetectionBoxes_Precision/mAP@.75IOU: 0.150158
DetectionBoxes_Precision/mAP (small): 0.012091
DetectionBoxes_Precision/mAP (medium): 0.200361
DetectionBoxes_Precision/mAP (large): 0.469002
DetectionBoxes_Recall/AR@1: 0.035397
DetectionBoxes_Recall/AR@10: 0.168582
DetectionBoxes_Recall/AR@100: 0.244226
DetectionBoxes_Recall/AR@100 (small): 0.066734
DetectionBoxes_Recall/AR@100 (medium): 0.312574
DetectionBoxes_Recall/AR@100 (large): 0.546237

Some inference samples are shown below. Looking at some inference examples, we can see that detections look fairly good for vehicles nearer in the image, and some farther objects are missed. This is shown more clearly in Figure 3.1-4. Another failure mode seen that is visible in Figure 3.1-5, are false positives or multiple detections of the same object instance. In this figure we can see that a red bush is falsely classified as a vehicle. Additionally, the vehicle on the right has 3 separate bounding box detections. These detections failed to be associated and filtered by non-max-suppression due to their iou being slightly below the 0.6 threshold used in the pipeline. In Figure 3.1-6 we see many false negatives, where many pedestrians are not detected, especially when in crowds. Pedestrians that are detected are often seen with low confidence scores, indicating that the

model is not well optimized for detecting pedestrians in general. This result matches what was seen earlier, where the class breakdown of the dataset heavily favours vehicles.

Figure 3.1-4: Reference inference sample image 1. This sample shows some missed detections of more distance vehicles.

Figure 3.1-5: Reference inference sample image 2. This sample shows some vehicle false positives and multiple detections of single a instance.

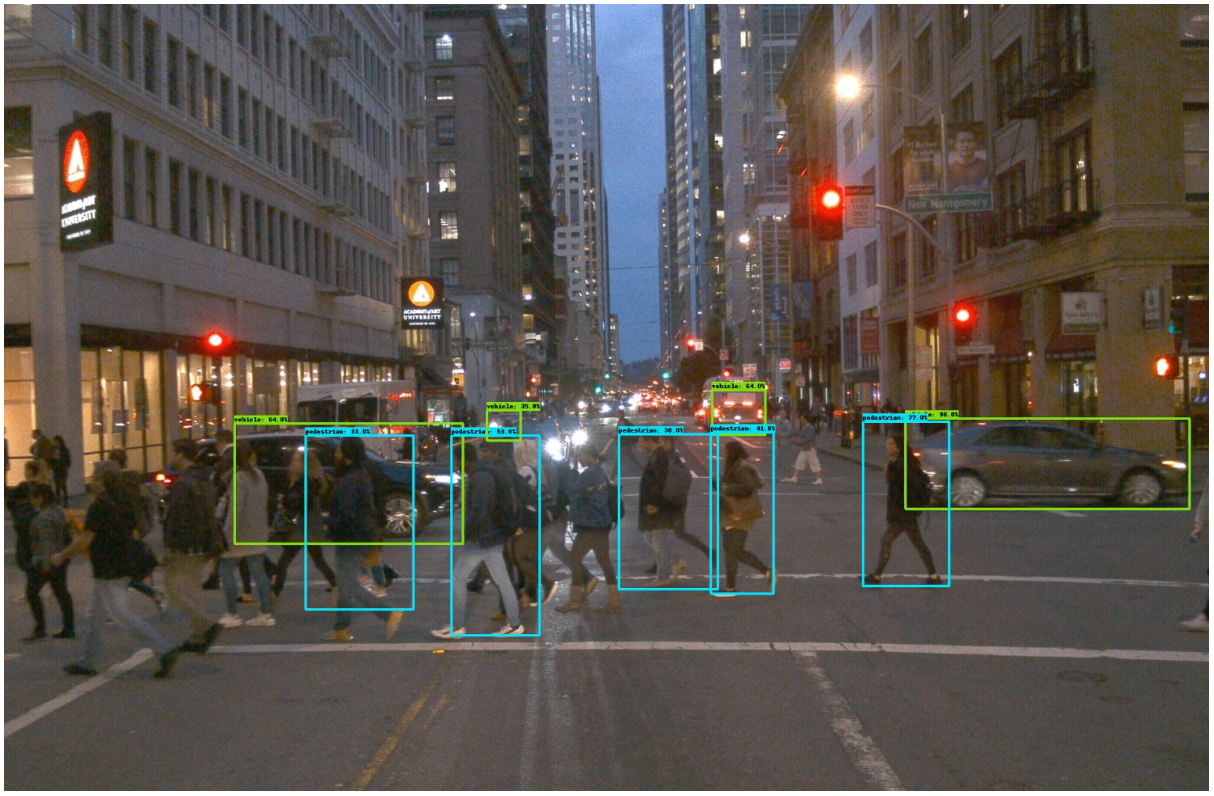

Figure 3.1-6: Reference inference sample image 3. This sample shows numerous false negatives where specifically for pedestrians, many instances are missed.

3.2 Improvements on the Reference

3.2.1 Augmentations

The primary aspect in which I looked to improve was in the data augmentations. The default configuration simply utilized random flipping and cropping, but not much in changing the appearance of images.

Color to gray augmentation was added to help provide more invariance to object and environment colors. For example, this should help promote that a vehicle should be detected based on its shape and texture appearance, and less so on the color of the vehicle.

Brightness augmentation was added to help provide better invariance to environmental lighting. This may include bright and sunny days or darker nights.

Contrast and saturation augmentations were added to provide further examples of distorted color and appearance to the scenes such that the model would not rely too heavily on the appearance of objects based on their typical colors.

Augmentations are shown below in images from the same tfrecord such that the augmentation changes are more obvious to see.

Figure 3.2.1-1: Grayscale augmentation.

Figure 3.2.1-2: Random adjust brightness.

Figure 3.2.1-3: Random adjust contrast.

Figure 3.2.1-4: Random adjust saturation.

3.2.2 Hyperparameter Improvements

Besides adding augmentations, I experimented with alternate batch sizes, iterations and optimizers, however many of those changes did not produce favourable results.

Ultimately the best performing improved model used a batch size of 8, trained iterations 50000 using the momentum optimizer with cosine decay. This model training config can be found in “**training/improved4**” in my github repo.

3.2.3 Improved Results

The improved model was trained using the above additional augmentations with a batch size of 8 for 50000 iterations. Training loss viewed in tensorboard during training can be seen below. The loss consistently drops throughout the course of training, again indicating that at least for the iterations performed, optimization did not get stuck in a local minima. Validation loss again follows a consistent downward trend throughout training indicating that overfitting has not likely occurred. Even after 50000 iterations, validation error still appears to be dropping, indicating that potentially training for more iterations could further improve performance.

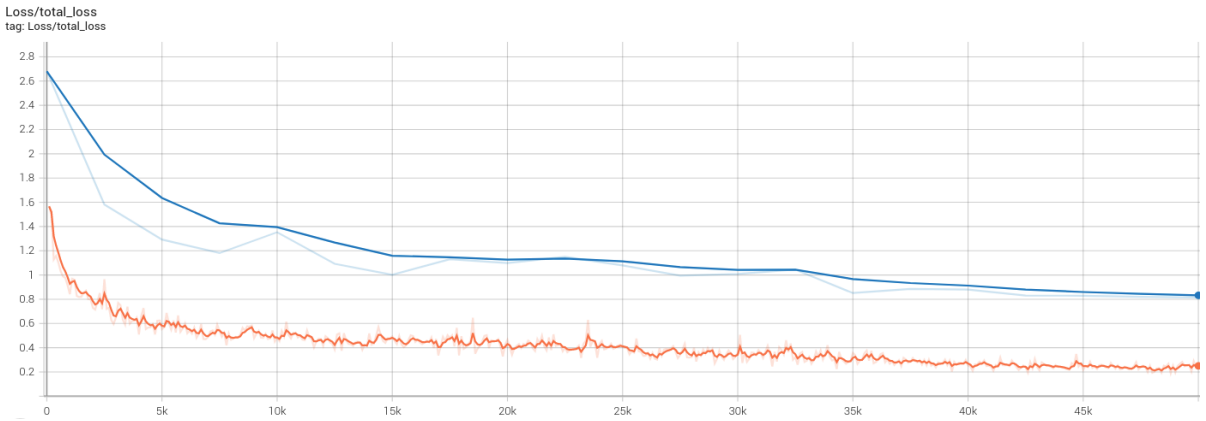

Figure 3.2.3-1: Improved model training and validation loss.

The validation metrics during training can be seen below. As was the case before, they steadily increase during training. Given that the metrics do not appear to have flattened out after all iterations, it is possible that further training could further improve performance.

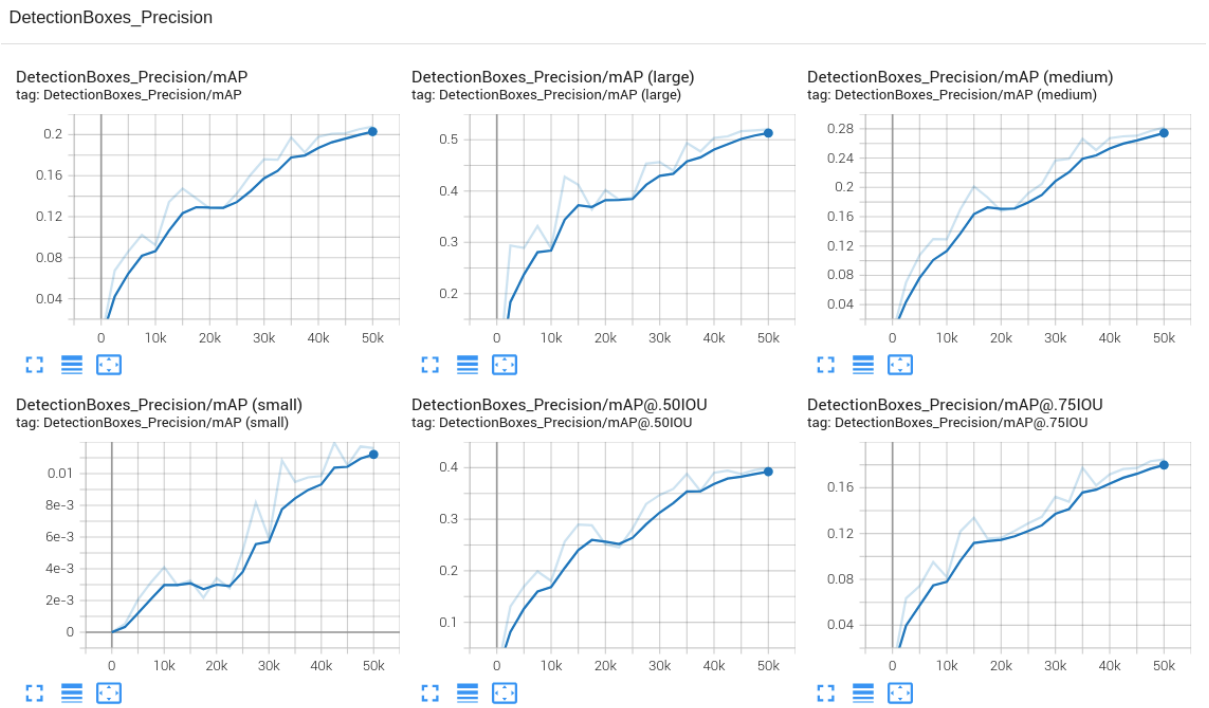

Figure 3.2.3-2: Improved model validation precision metrics during training.

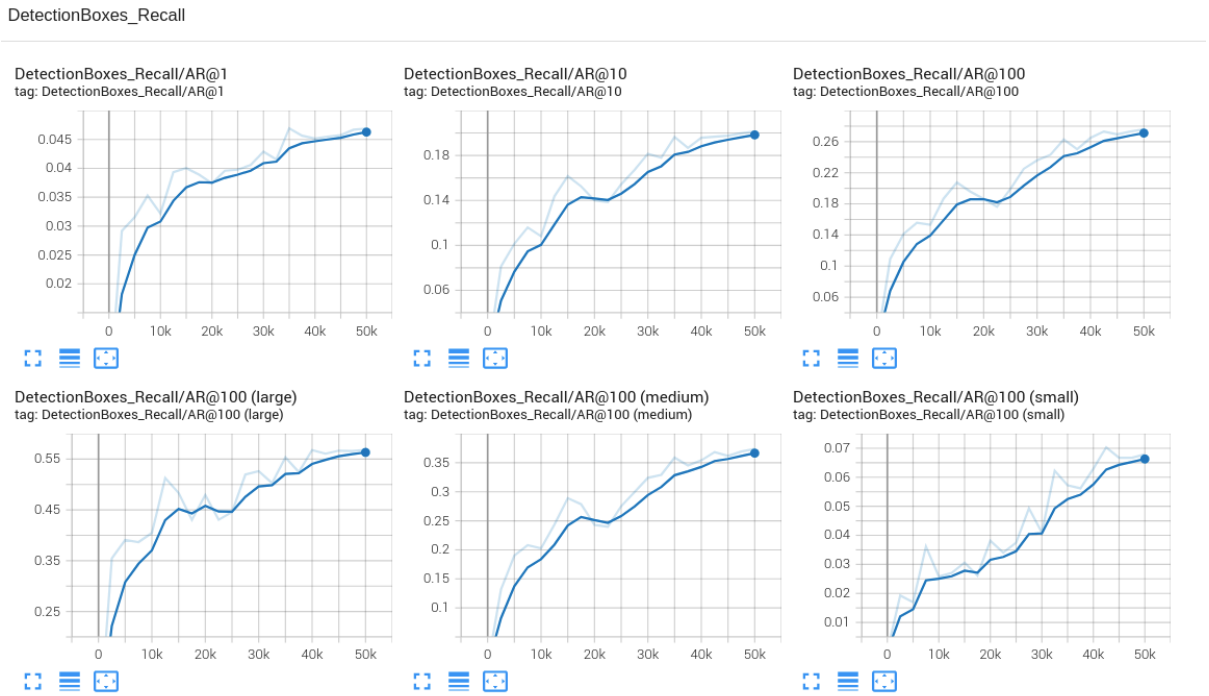

Figure 3.2.3-3: Improved model validation recall metrics during training.

The final evaluation on the Validation Dataset results are listed below:

DetectionBoxes_Precision/mAP: 0.207748
DetectionBoxes_Precision/mAP@.50IOU: 0.399413
DetectionBoxes_Precision/mAP@.75IOU: 0.184694
DetectionBoxes_Precision/mAP (small): 0.011609
DetectionBoxes_Precision/mAP (medium): 0.281783
DetectionBoxes_Precision/mAP (large): 0.520334
DetectionBoxes_Recall/AR@1: 0.046894
DetectionBoxes_Recall/AR@10: 0.201423
DetectionBoxes_Recall/AR@100: 0.275836
DetectionBoxes_Recall/AR@100 (small): 0.067851
DetectionBoxes_Recall/AR@100 (medium): 0.373761
DetectionBoxes_Recall/AR@100 (large): 0.568221

The final evaluation on Testing Dataset results are listed below:

DetectionBoxes_Precision/mAP: 0.173915
DetectionBoxes_Precision/mAP@.50IOU: 0.339627
DetectionBoxes_Precision/mAP@.75IOU: 0.150055
DetectionBoxes_Precision/mAP (small): 0.015082
DetectionBoxes_Precision/mAP (medium): 0.195966
DetectionBoxes_Precision/mAP (large): 0.483145
DetectionBoxes_Recall/AR@1: 0.035983

DetectionBoxes_Recall/AR@10: 0.169553
DetectionBoxes_Recall/AR@100: 0.239846
DetectionBoxes_Recall/AR@100 (small): 0.063549
DetectionBoxes_Recall/AR@100 (medium): 0.306214
DetectionBoxes_Recall/AR@100 (large): 0.546304

As can be seen there are small but significant improvements in the evaluation metrics across the board using the improved model training discussed in this section. The overall pattern of larger objects performing better than smaller ones is maintained. This makes sense because nearer objects are still more distinct and obvious than farther ones, and no special optimizations were done to weight smaller detections differently in the improved mode. This should be largely fine on a self driving car system as the nearer objects are the more relevant objects when navigating an urban environment. Only after the nearer objects are detected reliably, should focus shift the farther objects.

The same inference samples are shown below when put through this improved model. Looking at some inference examples, we can see that detections still look fairly good for vehicles nearer in the image, with fewer farther objects being missed.

In Figure 3.2.3-4, we can see a large improvement in detecting vehicles that are farther from the camera. Specifically the two farther back in the oncoming lane are now detected, which were not detected in the reference model. Figure 3.2.3-5 shows fewer false positive detections, where previously a red bush was detected as a vehicle and there were extra detections on the vehicle on the right side of the image. Figure 3.2.3-6 shows a significant improvement in the model's ability to detect pedestrians in a crowded environment. In the reference model, fewer than half of the walking pedestrians were detected, whereas in the improved model many more pedestrians are seen with a good detection.

Figure 3.2.3-4: Improved inference sample image 1. This sample shows improved results detecting more distant vehicles.

Figure 3.2.3-5: Improved inference sample image 2. This sample shows improved results with fewer false positive detections and multiple detections of a single object instance.

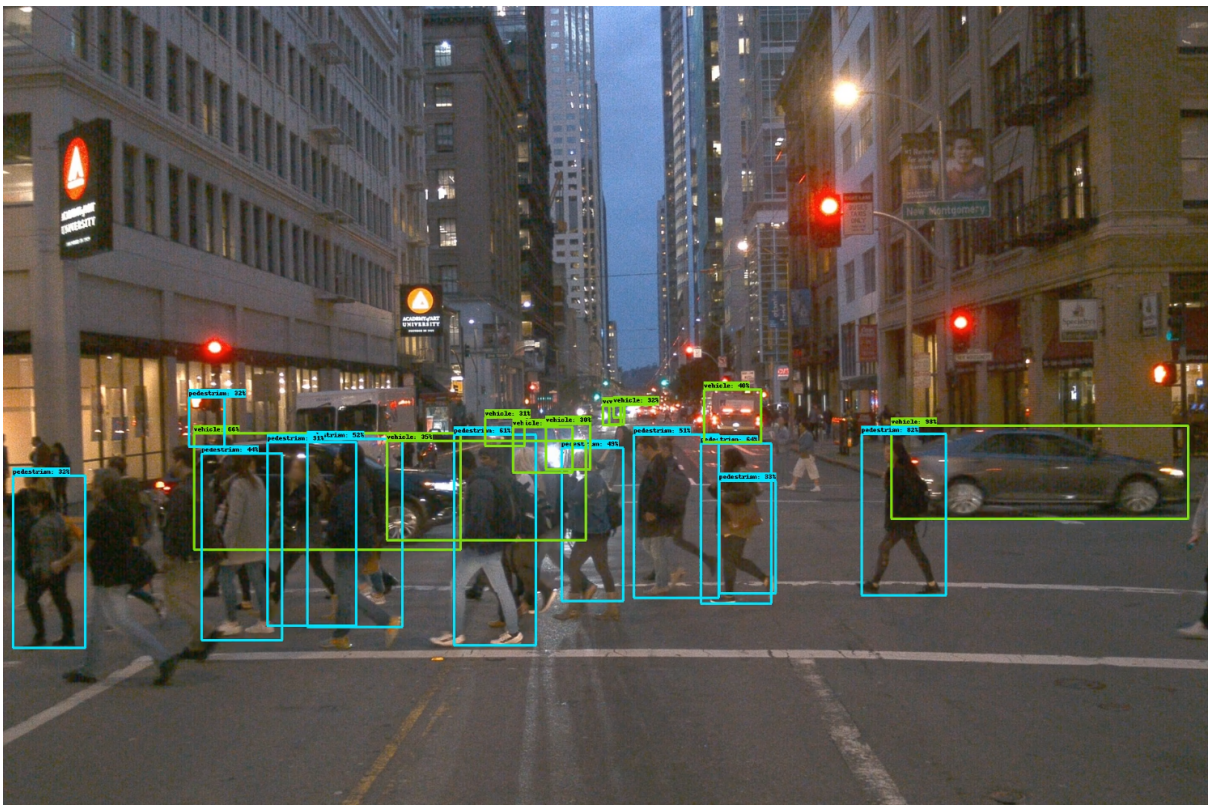

Figure 3.2.3-6: Improved inference sample image 3. This sample show improved results in detecting pedestrians in a crowded environment.

4 Conclusion

In this project we explored deep learning object detection model training for urban environments. This project utilized real world data from the waymo dataset to provide a representative problem of what might be seen when looking to develop self-driving cars in the real world. It was shown that models can perform quite differently depending on how their input data and optimization parameters are tuned. In my case, a couple additional augmentation, with extended training iterations was found to create an improved model with significantly better detection performance than a reference model.

Even with my improvements, I believe there is still much more than could be done. There are further explorations of additional augmentations as well as different object detection model designs and optimization parameters that may be found to perform even better.