# Object Detection in an Urban Environment

## Table of Contents



# Project Overview

In this project we are using data from the Waymo open dataset in order to train an object detection algorithm that is capable of detecting vehicles, pedestrians and cyclists solely from camera data and providing a boundary box for them. In order to achieve this 100 segments are downloaded from the Waymo dataset, preprocessed, analyzed, splitted in suitable train, validation and test sets and used for training of a neural network. After a first iteration the detection performance is evaluated and measures are taken with the goal of improving it. After using them for further training the performance of the final model is validated on the test dataset.

The object detection is a crucial component of self driving vehicles because we can gather important information about the surroundings of the car that can be used as input for several control functions e.g.:

- automatic lane change (check if target lane is free)
- collision avoidance (check if object is in path of ego vehicle)
- emergency braking
- general trajectory planning
- following a front vehicle (platooning)
- ...

# Setup

I struggled a lot with the provided Dockerfile and saw a bunch of unspecific Tensorflow Import errors after building it. After some efforts in debugging I decided to start from scratch and created a Dockerfile on my own, see Github.

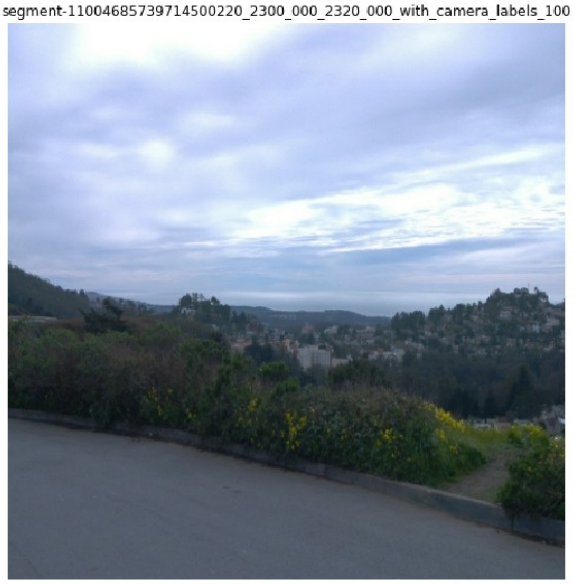
# Dataset

# Analysis

This sections describes some general observations on the available data that are retrieved from EDA. It is crucial to get an understanding of the used data to be able to interpret the final results and to solve possible problems.

## General

- There are frames without any visible object

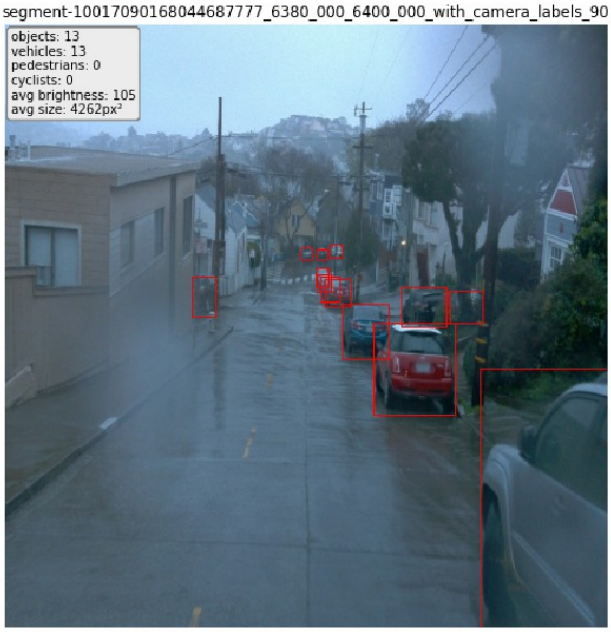



- We have high density of labeled objects, even if they are barely visible in the image (left). Trailers are labeled as separate objects (right).


## Light and Weather

- The are images with glare and high dynamic range (sky vs. shadow areas) and high overlap of objects. Most of the dataset was recorded under dry weather conditions


- There are some rare day and rain segments which are blurry and partially occluded with rain drops in front of the camera.


-

- There are also night scenes with low brightness and blurry images (left) and wet streets (right)

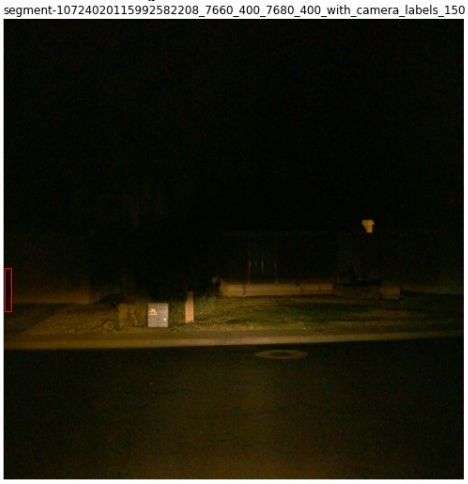

- Especially at night there are reflections


- Daytime glare does not seem to be a problem.

## Data quality

This section lists some characteristics of the dataset that might be problematic for training.

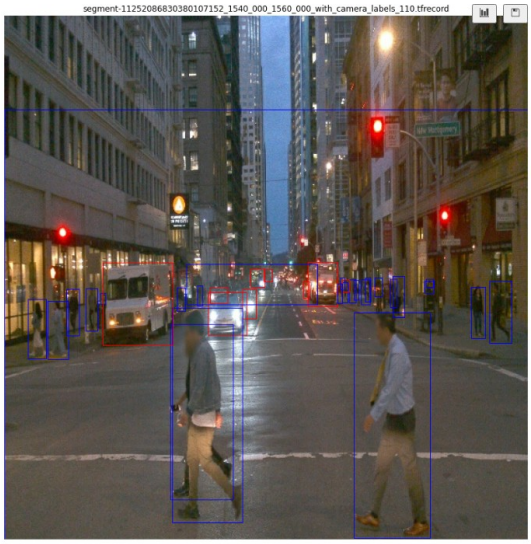
- There are images without visible objects but with annotations




- Most of the attributes in the *.tfrecord files are not filled with actual values after the preprocessing. However they are not used in the project therefor it is no problem.

```
image                              3d tensor
source_id                          segment-12012663867578114640_820_000_840_000_with_camera_labels_170.tfrecord
key
filename                           segment-12012663867578114640_820_000_840_000_with_camera_labels_170.tfrecord
groundtruth_image_confidences      []
groundtruth_verified_neg_classes   []
groundtruth_not_exhaustive_classes []
groundtruth_boxes                  2d tensor
groundtruth_area                   []
groundtruth_is_crowd               []
groundtruth_difficult              []
groundtruth_group_of               []
groundtruth_weights                [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1.]
groundtruth_classes                [1 1 2 1 1 2 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1]
groundtruth_image_classes          []
original_image_spatial_shape       [640 640]
```

- There are phantom annotations (big blue box = pedestrian)

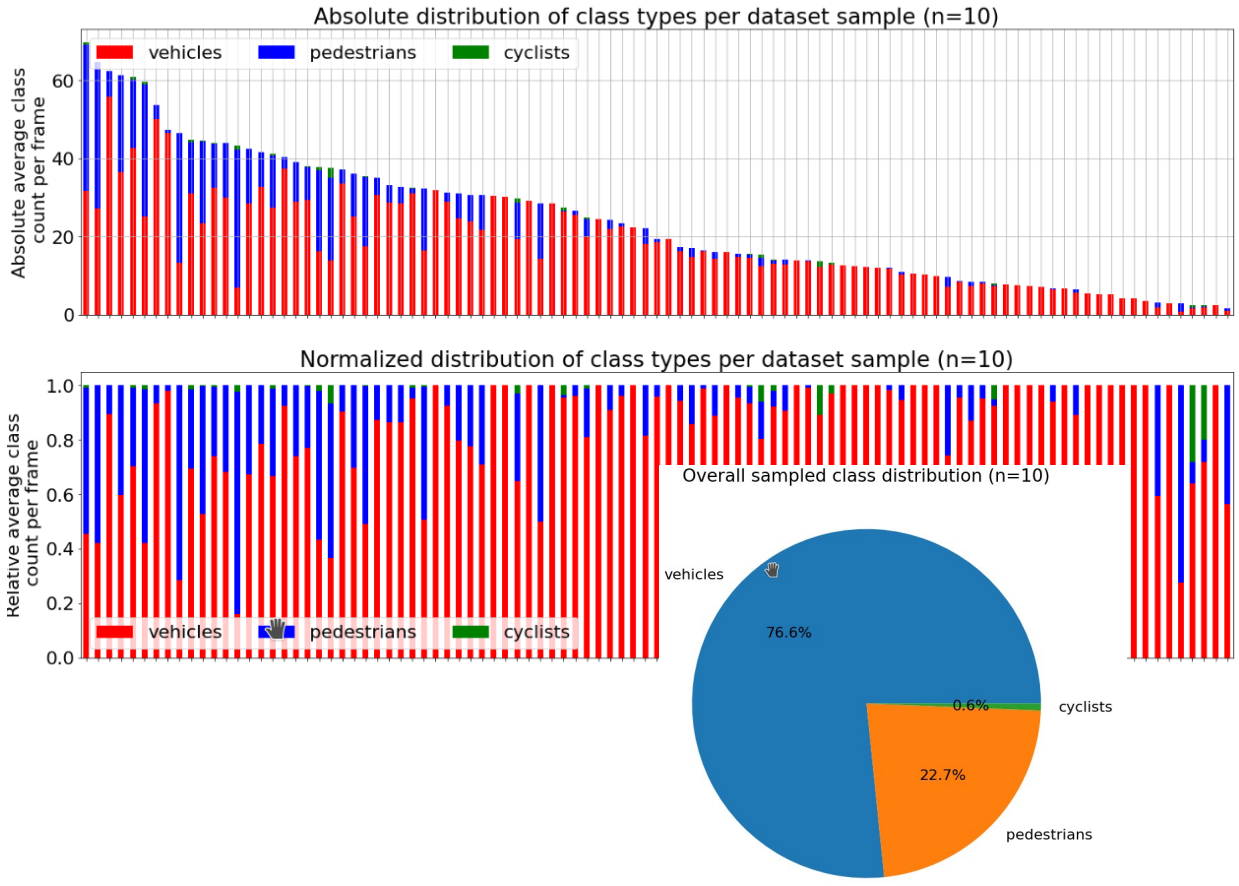



- some objects are not labeled (manually marked in purple)

- annotations are also sometimes wrong (portable toilet is labeled as vehicle on the left, pedestrian as cyclist on the right)



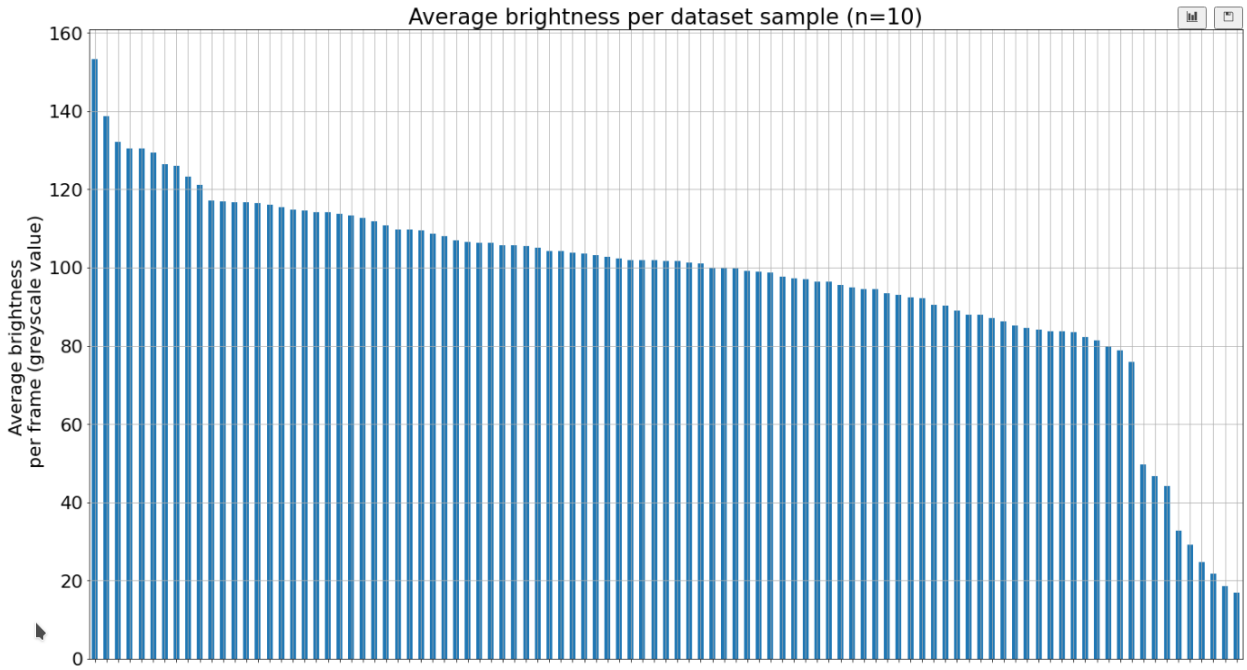
- The dataset is heavily unbalanced with respect to the class distribution. Roughly 76% of the objects are vehicles, 23% pedestrians and there are only very few cyclists (<1%).
  - These plots we created by picking 10 random frames from each *.tfrecord file and calculating the average count of each object over them.
  - There are segments in which pedestrians dominate, but in most files vehicles are the dominant class.
  - Side note: There are segments with very high numbers of objects (urban areas) visible at the same time and also segments with much less (suburbs)




- There are night scenes. I tried to categorize them by analyzing the average brightness of sampled frames from each segment. Therefor I converted these frames to grayscale and

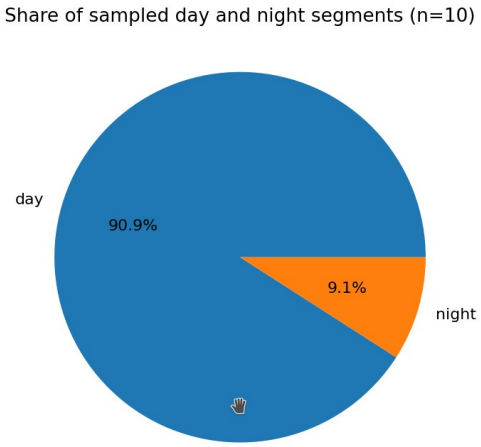
calculated the average pixel value. The following figure shows the results of each input file in descending order.

- It is obvious that there is a sharp decline for the right most files.
- Therefor a brightness threshold of 60 can be derived (lower values are considered as night scenes)




- Plotting them shows that they are indeed night scenes. This image shows two frames from the brightest and darkest segments each.

- After this day/night classification we can see that over 90% of the segments were recorded at day.




## Cross validation

In order to perform a meaningful cross validation we must split the available data in a training, validation and test batch. The goal for the split creation is to have datasets with similar properties regarding their share of day/night and object type share.

Following the cross validation lecture from the course I chose to take 75% of the available data for training, 15% for validation and 10% for the test dataset. Furthermore I made sure to pick equal amounts of night and day scenes for all of the three datasets in order to be balanced here.

The validation set will be used to optimize the model parameters and must be kept separate from the training dataset in order to be able to detect when the model starts overfitting (training loss keeps sinking, but val loss starts raising again). Test dataset will be used for final evaluation after the cross validation is done. I choose this approach in order to prevent ‘leaking’ of information from the validation dataset to the model.

# Training

Notes on starting and validating the training in my Docker:

I had to reduce the batch size in order not to run into memory issues on my hardware and move validation to the CPU only, see my post: https://knowledge.udacity.com/questions/807430

Create a config with

```
python /app/project/edit_config.py --train_dir /app/project/data/splitted/train --
eval_dir /app/project/data/splitted/val --batch_size 4 --checkpoint
/app/project/training/pretrained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/
checkpoint/ckpt-0 --label_map /app/project/label_map.pbtxt
```

Run the training with:

```
python /app/project/experiments/model_main_tf2.py
--model_dir=/app/project/training/reference/
--pipeline_config_path=/app/project/training/reference/pipeline_new.config
```

Run the validation only on CPU to avoid memory issues:

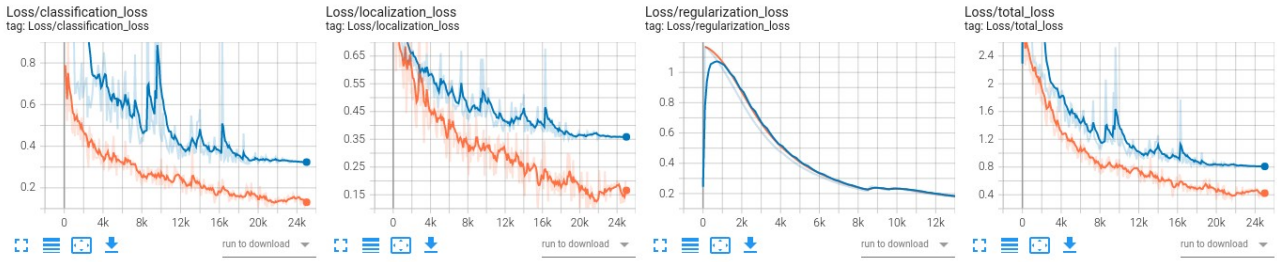
```
export CUDA_VISIBLE_DEVICES=-1

python /app/project/experiments/model_main_tf2.py
--model_dir=/app/project/training/reference/
--pipeline_config_path=/app/project/training/reference/pipeline_new.config --
checkpoint_dir=/app/project/training/reference/
```

Run tensorboard with

```
tensorboard --logdir=/app/project/training
```

## Reference experiment

After training for 25.000 epochs we get the following results (training in orange, validation in blue):

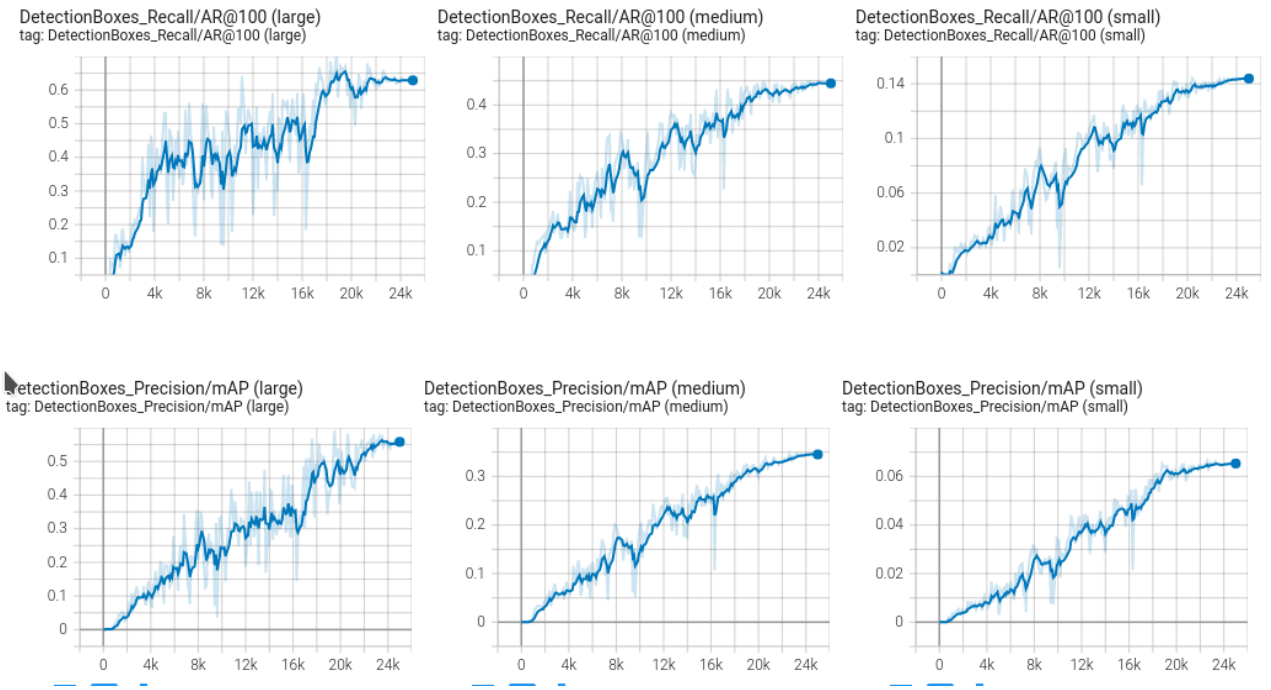



As can be seen the loss is successfully reduced during the training run, for both the train and validation dataset. However after 25.000 epochs train and val loss are still moving in parallel with a significantly higher val loss than the training loss. While both show a converging behavior further improvements can be expected by continuing the training process as no raising validation loss (would be a sign of overfitting) is observable.

The total loss is comprised of nearly equal parts of classification, localization and regularization loss. Therefor I think the weights of the loss types in the config do not have to be adjusted.

Looking at the actual frame based results it can be seen that the detection of vehicles in the near field works pretty decent. However vehicles far away are often missed. Also the detection of pedestrians performs very poorly.


This assessment is supported by looking at the recall and mAP charts for the validation data:



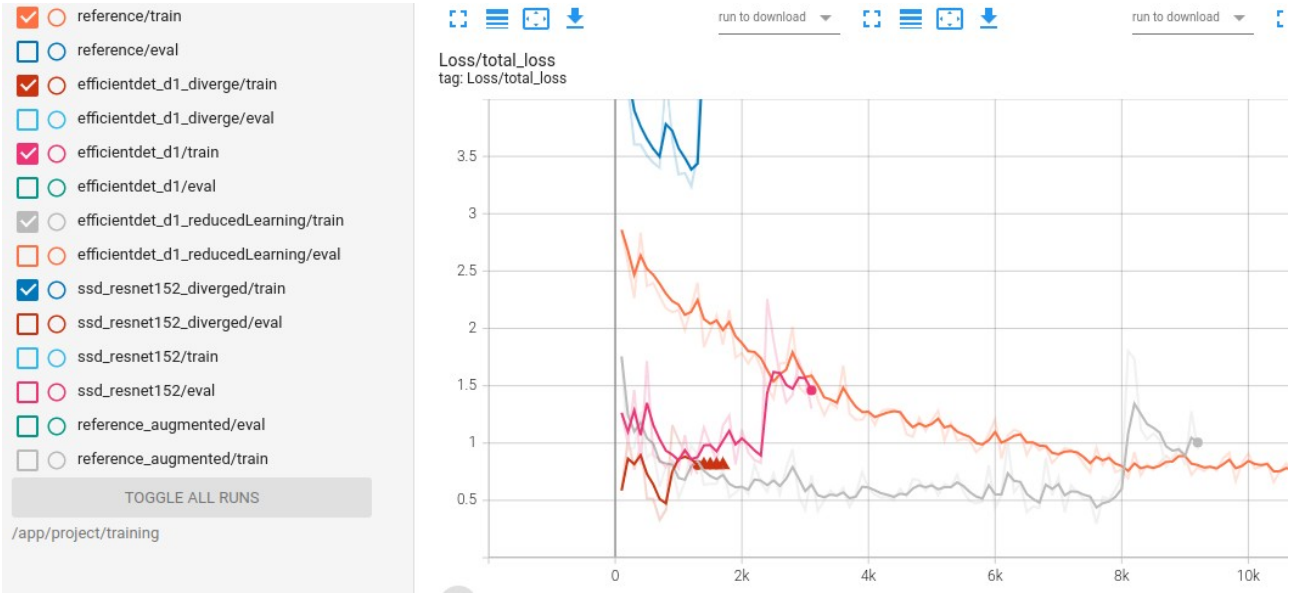

While the recall of large objects lays within a decent range after half of the training run, the value of mid ans small sized objects is much lower and only increases slowly. The recall for large objects also reaches a plateau while the others continue raising until the end of the training.

Regarding the mAP objects of all size seem to see improvements over the whole training run.

## Improve of the reference

Before trying out augmentation strategies and hyper parameter tuning I wanted to check the performance with another DNN from the TF object detection zoo. Because of a considerably higher mAP value on the COCO dataset (38.4 vs 34.3) I tried out EfficientDet D1 640x640. For using this model I had to reduce the batch size on my system.

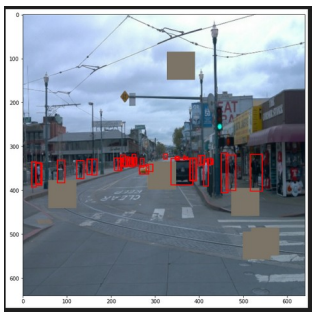
During training with mostly using the models default configuration values I ran into the problem, that after 1.800 epochs the metrics started getting NaN (dark red curve in figure below). After a bit of research I found out that this is related to divergence of the SGD algo. Therefor I reduced the learning rate to the same value as in the reference training run. Even after this adjustment I had the problem of suddenly exploding loss after several thousand iterations (pink and grey curves). I except the small batch size (n=2) is the origin of this problem because this makes the gradient during training very noisy. I found a similar behavior after trying out SSD ResNet152 V1 FPN 640x640 (blue curve).



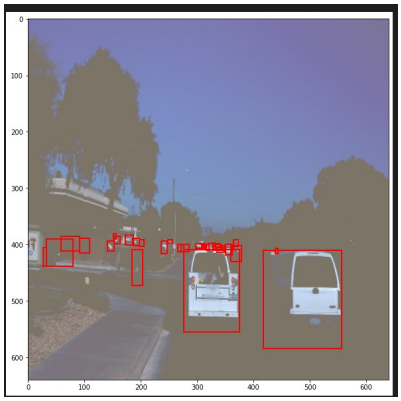

After these experiences I decided to stick to the reference ResNet50 and use some augmentations in order to try to improve the performance. After playing around with the provided Jupyter Notebook I found the following to be meaningful

- random_horizontal_flip which should help on the generalization

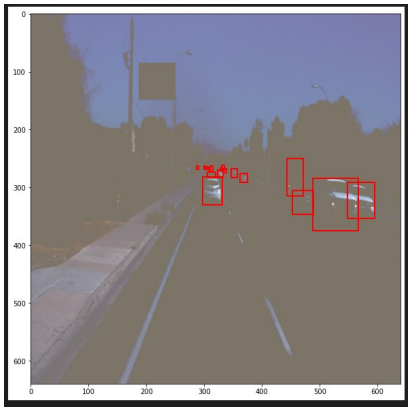
- random_black_patches(probability=0.25). The intent here is to improve the detection of partially covered objects. Note for some reasons the rendering in the jupyter notebook displays the black patches as brown. Even if this behavior should be the same during the training it will most probably not hurt, because the whole point of the patches is to hide objects and structures in the image an not to influence the color.



- random_adjust_brightness(mag_delta=0.02) This should improve the detection at night (as the dataset contains only few night scenes)



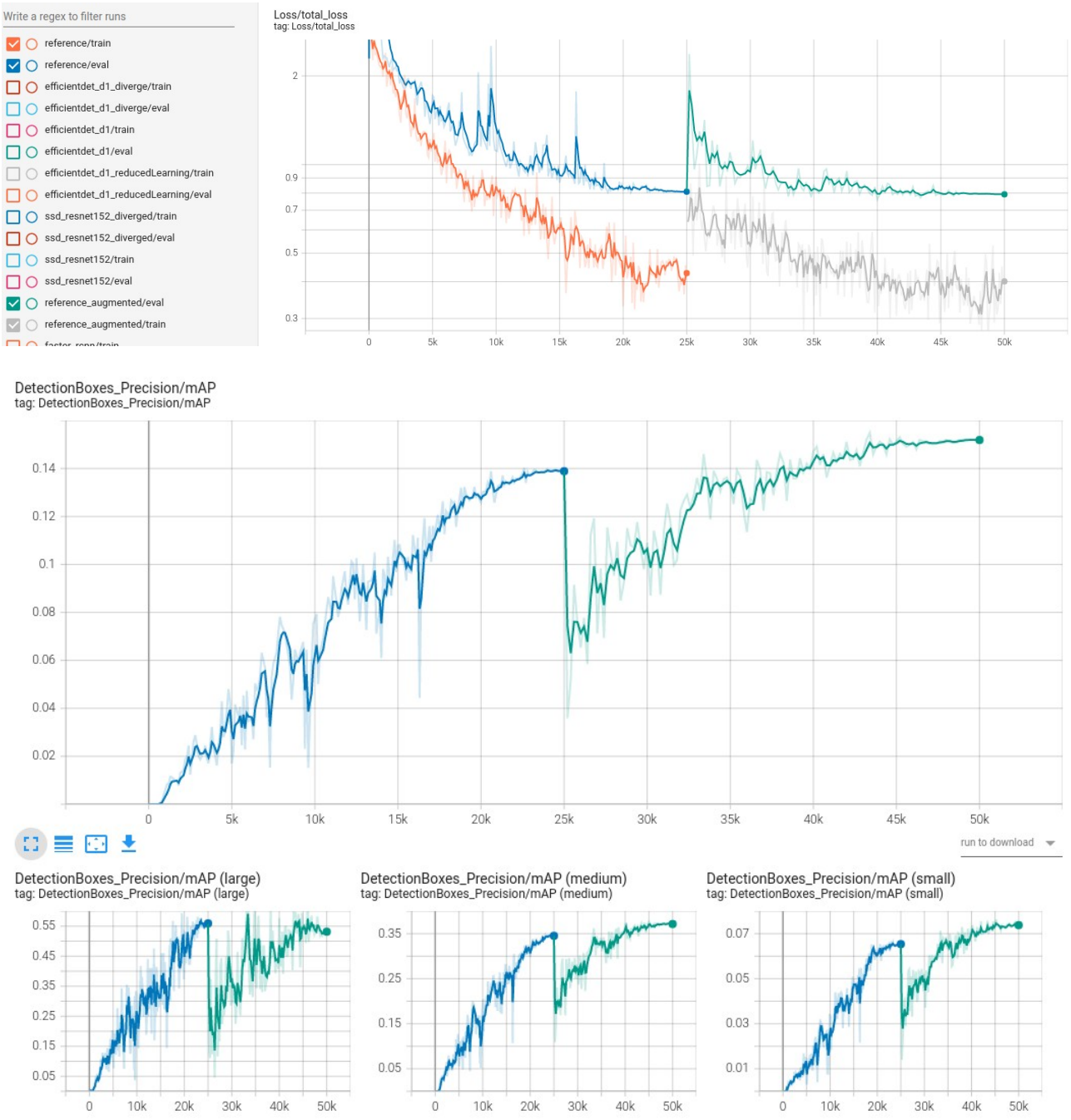
- An example of these combined augmentations is provided below:



Furthermore I decided to continue using the random crop image augmentation, that was already part of the reference run.

Because the reference training did not show signs of overfitting I will use the last training checkpoint from this run to start the training with the augmented dataset for 25.000 more steps.

The results are shown in the plots below. It can be seen, that the mAP significantly improved by roughly 9% from 0.139 (blue reference) to 0.152 (green run with augmentations) despite the fact that the overall validation loss is practically identical what was somehow surprising for me. Another point that I did not expect was the sharp rise in loss after continuing the training with my previous reference checkpoint. This is probably because of the additional augmentations. Looking at the breakdown of the different object sizes it can be seen, that by augmenting the data especially the detection of small and mid sized objects got improved while large object detection even slightly dropped. There seems to be a trade-off here.




After this analysis I was satisfied with the cross validation results and exported the model with

```
python /app/project/experiments/exporter_main_v2.py --input_type image_tensor --
pipeline_config_path /app/project/training/reference_augmented/pipeline_new.config --
trained_checkpoint_dir /app/project/training/reference_augmented --output_directory
/app/project/training/reference_augmented/export
```

and finally used the test dataset to generate an inference video

```
python /app/project/inference_video.py --labelmap_path /app/project/label_map.pbtxt --
model_path /app/project/training/reference_augmented/export/saved_model/ --
tf_record_path /app/project/data/splitted/test/segment-
10391312872392849784_4099_400_4119_400_with_camera_labels.tfrecord --
config_path /app/project/training/reference_augmented/pipeline_new.config --
output_path  /app/project/data/splitted/test/segment-
10391312872392849784_4099_400_4119_400_with_camera_labels.mp4
```

The results are similar to the analysis on single frame level: