Report:

1- Explanatory Data Analysis:

As mentioned in the lectures, the main and first step in training is to understand the data. It is important that the dataset is enough rich and includes many possible driving situations with different road users (e.g. pedestrians, cyclists, other vehicles, …) involved. The dataset should be balanced and unbiased to have a fair model. To analyse the dataset, first, I wrote a function which takes a batch of dataset as an input and display an image with its corresponding bounding boxes (vehicles in red, pedestrians in blue, cyclist in green).

Here is the function:

==============================

```
def display_instances(batch):
    """
    This function takes a batch from the dataset and display the image with
    the associated bounding boxes.
    """
    color_map = {1: 'red', 2: 'blue', 4: 'green'}
    fig, ax = plt.subplots(figsize=(15,15))
    ax.axis('off')
    image = batch['image'].numpy()
    ax.imshow(image)
    height, width = image.shape[:2]
    boxes = batch['groundtruth_boxes'].numpy()
    classes = batch['groundtruth_classes'].numpy()
    for bbox, cl in zip(boxes, classes):
        y1, x1, y2, x2 = bbox
        y1 *= height
        x1 *= width
        y2 *= height
        x2 *= width

        bb_width = x2 - x1
        bb_height = y2 - y1
        box = Rectangle((x1, y1), x2-x1, y2-y1, facecolor='none', edgecolor=color_map[cl])
        ax.add_patch(box)

    plt.show()

    return
```

============================

The images below show the samples of the batch as the output of the function. The initial and primary analysis shows that the number of cyclist in the datasets are much lower than the number of vehicles and pedestrians which can impact the model performance in detecting cyclist.

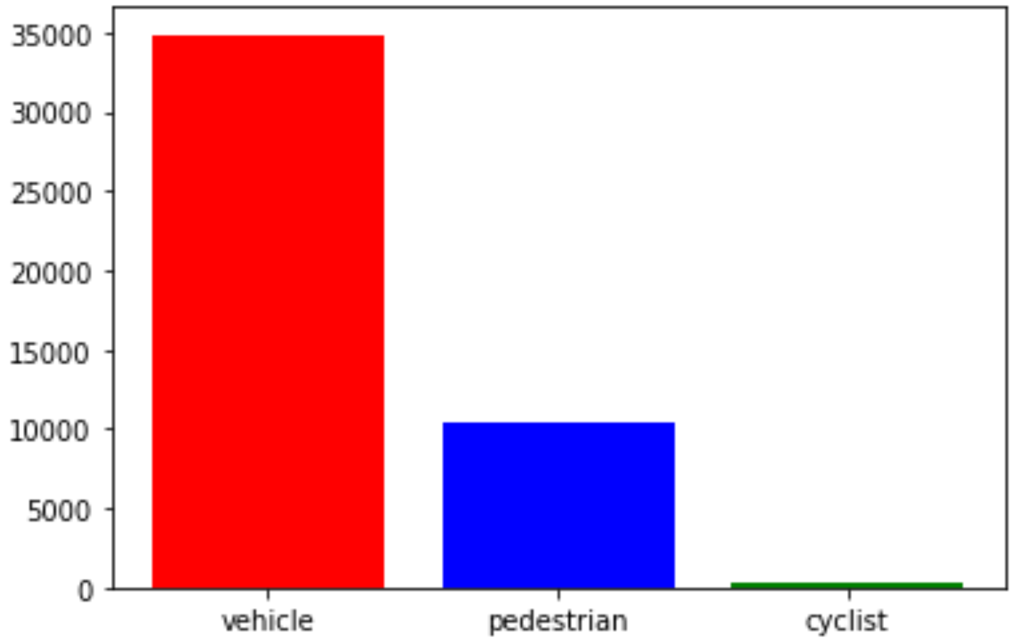
In order to analyse the dataset in details the below code is developed to extract more information from the dataset.

```
import glob
import sys
import tensorflow as tf

def dataset_analysis(record):
   return tf.io.parse_single_example(
      record,


     {
      "image/height": tf.io.FixedLenFeature([], tf.int64, default_value=0),
      "image/width": tf.io.FixedLenFeature([], tf.int64, default_value=0),
      "image/filename": tf.io.FixedLenFeature([], tf.string, default_value=""),
      "image/source_id": tf.io.FixedLenFeature([], tf.string, default_value=""),
      "image/encoded": tf.io.FixedLenFeature([], tf.string, default_value=""),
      "image/format": tf.io.FixedLenFeature([], tf.string, default_value=""),
      "image/object/bbox/xmin": tf.io.VarLenFeature(tf.float32),
      "image/object/bbox/xmax": tf.io.VarLenFeature(tf.float32),
      "image/object/bbox/ymin": tf.io.VarLenFeature(tf.float32),
      "image/object/bbox/ymax": tf.io.VarLenFeature(tf.float32),
      "image/object/class/text": tf.io.VarLenFeature(tf.string),
      "image/object/class/label": tf.io.VarLenFeature(tf.int64)
     }
   )

dataset = tf.data.TFRecordDataset(glob.glob('/home/workspace/data/waymo/training_and_validation_copy/*.tfrecord')).map(dataset_analysis)

class_count = {1: 0, 2 : 0, 4: 0}
class_dist = {1:[], 2:[], 4:[], 'total':[]}
min_all = sys.maxsize
```

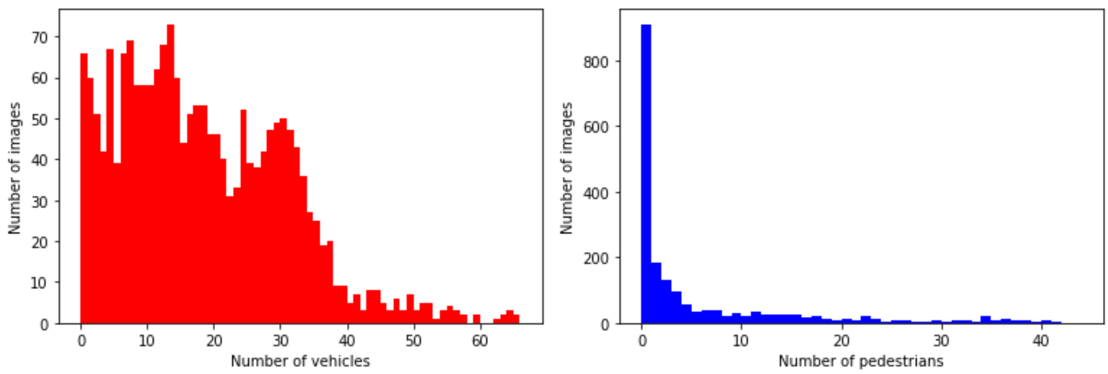
```
min_all_batch = None
max_all = -1
max_all_batch = None
min_vehicle = sys.maxsize
min_vehicle_batch = None
max_vehicle = -1
max_vehicle_batch = None
min_pedestrian = sys.maxsize
min_pedestrian_batch = None
max_pedestrian = -1
max_pedestrian_batch = None
min_cyclist = sys.maxsize
min_cyclist_batch = None
max_cyclist = -1
max_cyclist_batch = None

for batch in dataset.take(-1):
    count = {1: 0, 2 : 0, 4: 0 }
    total = 0
    for cl in tf.sparse.to_dense(batch["image/object/class/label"]).numpy():
        total += 1
        class_count[cl] += 1
        count[cl] += 1
    class_dist[1].append(count[1])
    class_dist[2].append(count[2])
    class_dist[4].append(count[4])
    class_dist['total'].append(total)

    if total < min_all:
        min_all = total
        min_all_batch = batch
    if total > max_all:
        max_all = total
        max_all_batch = batch
    if count[1] < min_vehicle:
        min_vehicle = count[1]
        min_vehicle_batch = batch
    if count[1] > max_vehicle:
        max_vehicle = count[1]
        max_vehicle_batch = batch
    if count[2] < min_pedestrian:
        min_pedestrian = count[2]
        min_pedestrian_batch = batch
    if count[2] > max_pedestrian:
        max_pedestrian = count[2]
        max_pedestrian_batch = batch
    if count[4] < min_cyclist:
```

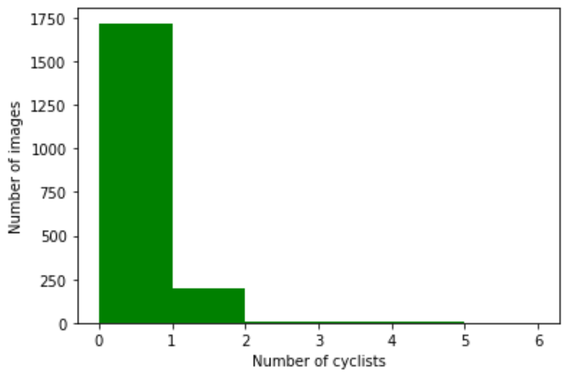
```
        min_cyclist = count[4]
        min_cyclist_batch = batch
    if count[4] > max_cyclist:
        max_cyclist = count[4]
        max_cyclist_batch = batch
```

The analysis of the dataset shows that there are **1937** images in the dataset. The figure below shows the distribution of classes in the dataset. As shown in the figure, the dataset is not balanced and it includes very small number of cyclists compared to pedestrians and vehicles.



Figures below show the number of vehicles, pedestrians and cyclists in images:





2- Data split:

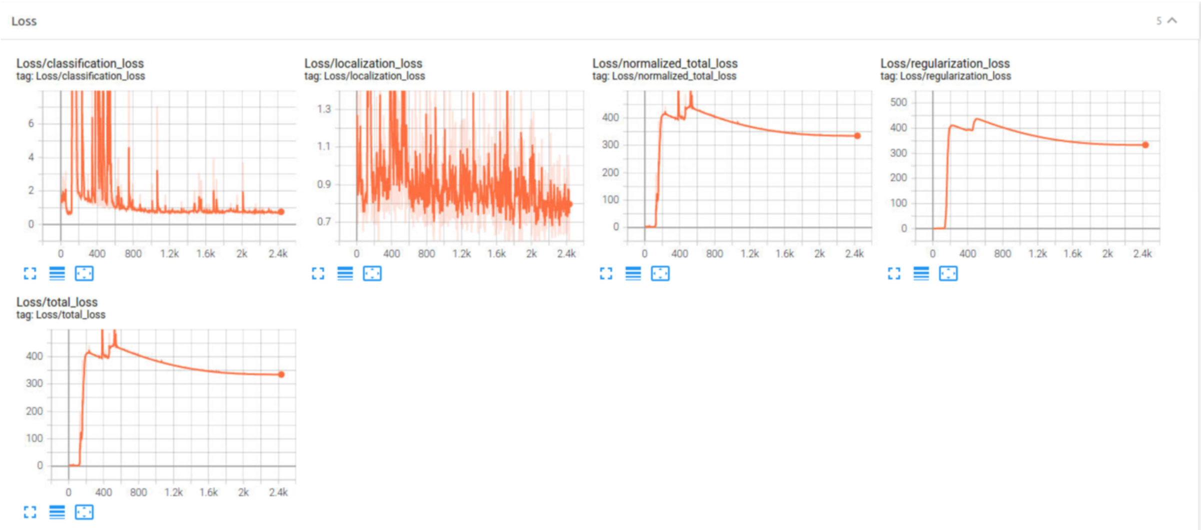
In next step, I developed the code to split the data provided in project folder into training and validation datasets. To split the data, I used the ratio 80% (for training) and 20% (for validation) but since the dataset is small, it may be better to split it as 75%-25%. But it is noted that it is important to shuffle the dataset in order to avoid any potential correlation between data. To split the data, the below function has been written:

```
def split(data_dir):
   """
   Create three splits from the processed records. The files should be moved to new folders
in the
   same directory. This folder should be named train, val and test.

   args:
      - data_dir [str]: data directory, /home/workspace/data/waymo
   """
   dataset = os.listdir(data_dir + '/training_and_validation/')
   random.shuffle(dataset)

   dataset_size = len(dataset)
   training_size = dataset_size * 0.8
   validation_size = dataset_size * 0.2

   for i, tfrecord in enumerate(dataset):
      if i<training_size:
         shutil.move(data_dir + '/training_and_validation/' + tfrecord, data_dir + '/train')

      else:
         shutil.move(data_dir + '/training_and_validation/' + tfrecord, data_dir + '/val')
```

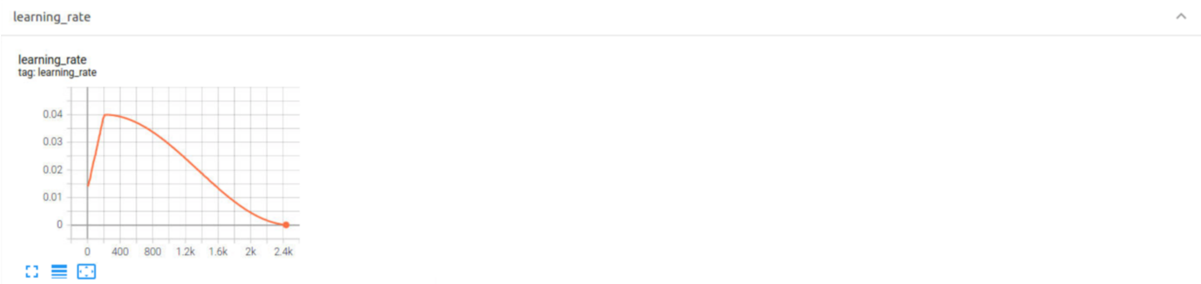
3- Model training and evaluation:

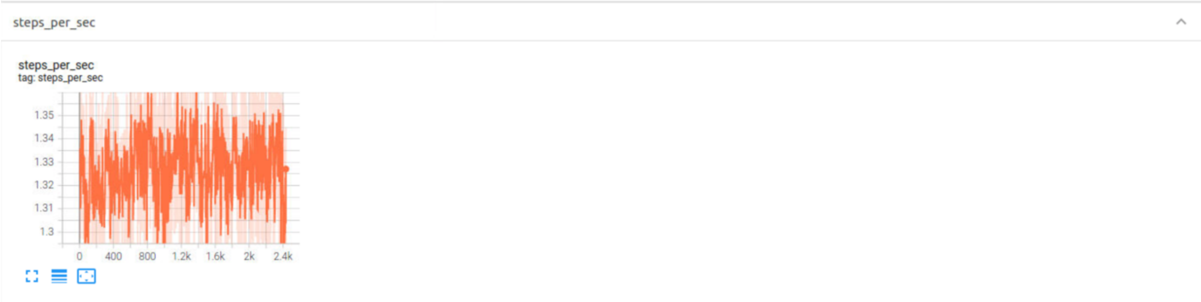
In this step, I trained the model using the training dataset and evaluated the model performance using evaluation dataset. The results below show the training and evaluation performances.

The figure below shows the loss values during training. As can be seen, the model learns but with not a great performance.

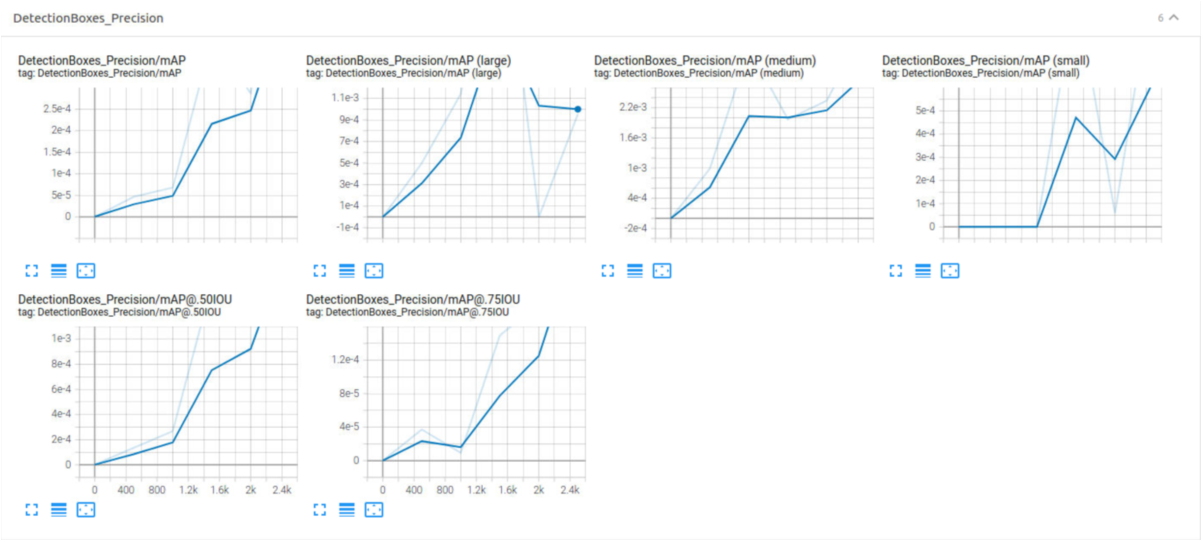


A variable learning rate has been used to make the learning process efficient. At early Epochs, the learning rate increases to warm up the model and after that with increase in epochs number the learning rate reduces.





The figures below show the precision and recall metrics. As can be seen while model learns both precision and recall increase as well., representing the improvement in model performance.



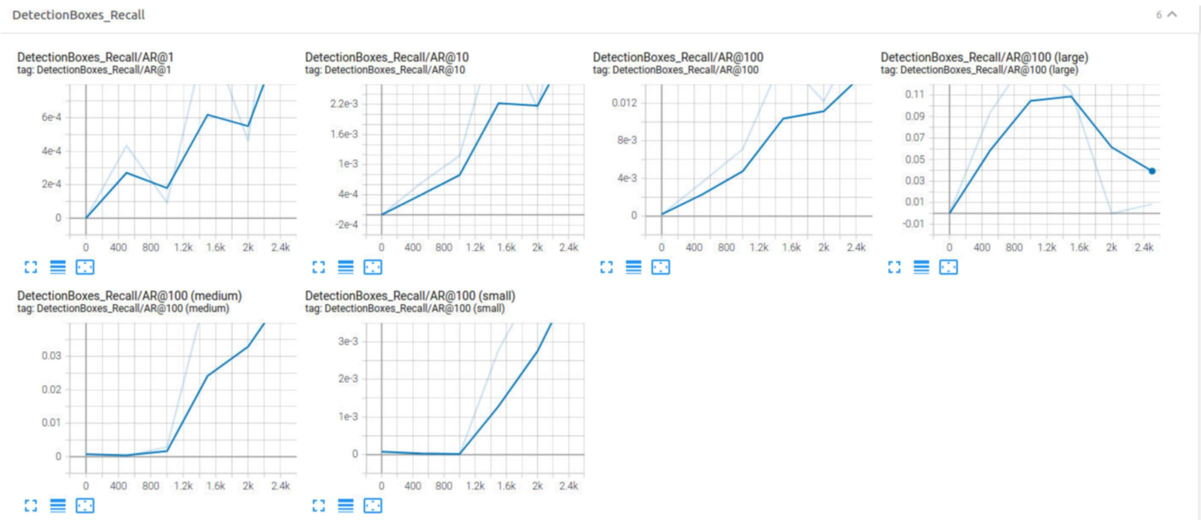
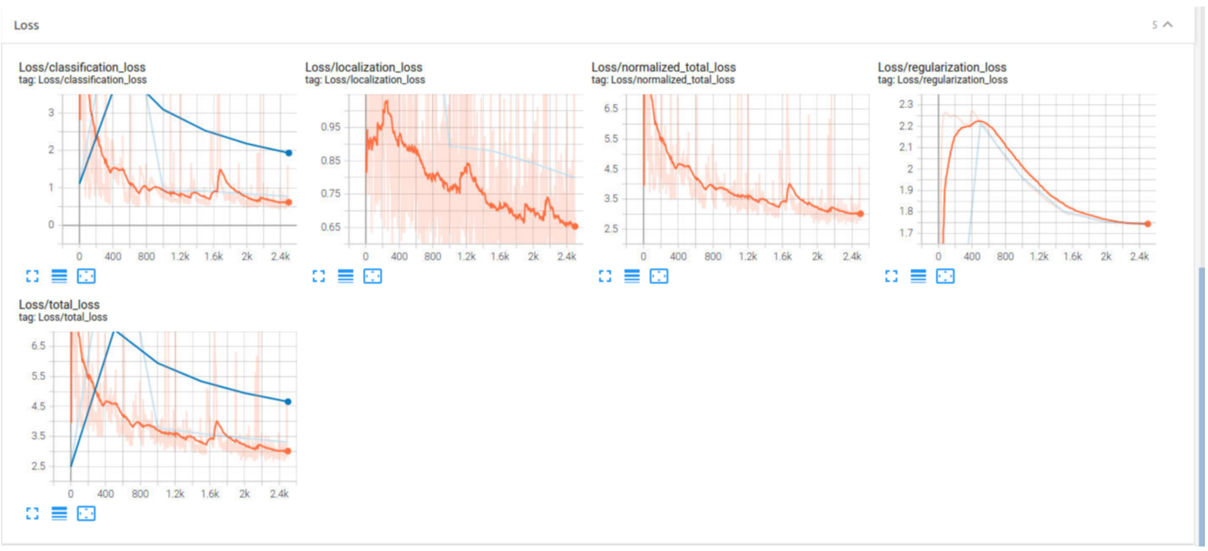


Figure below compares the loss values of training and evaluation process. As the results show the model has not been overfitted.



4- Performance improvement:

The performance can be improved by fine tuning of the hyperparameters, such as the number of batches, learning rate, activation function etc. Also, one way of improvement is to enrich dataset through data augmentation. Unfortunately, the provided data augmentation file was damaged and I couldn’t run it for further analysis.