

Project : Object Detection in an Urban Environment

Project Overview

In this project,

On the Early stage of project, I tried to setup a local environment on my laptop. But , I been faced so many errors(version not match each other between the packages), finally I gave up to build a local environment. (it tooks 11 days!!). So, I asked you a environment.yml files for the project.)

Finally, Project is developed under the udacity workspace environment.

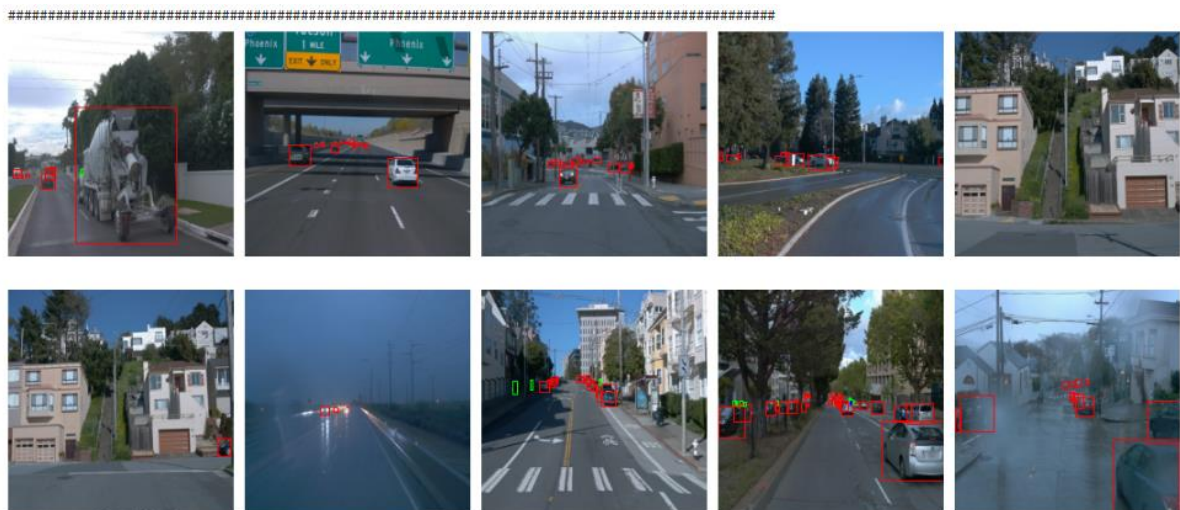
Python package is well match each other only on the Udacity workspace Environment!

The first stage is **the explore the dataset**.

This ipynb is to give a motivation for the students to explore and study dataset.

What is contained on the dataset(trfrecord).

With using display images and dataset_choice functions together,. Finally result is shown below.



These images with bounding boxes with using tensorflow and

waymo_open_dataset(dataset_pb2),matplotlib to indicate the vehicles on the images.,

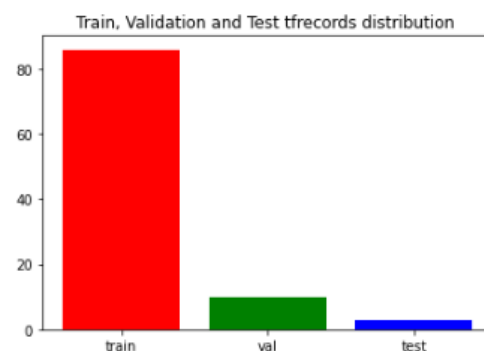
The next step is EDA chapter.

The dataset show train,val,test set's distributions. Absolutely, train dataset is the big one.

With using class_count function, clearly recognize the distributions of 3 classes(vehicle, pedestrian,cyclist) for each train and test dataset. On the train dataset, pedestrian data is much bigger(2.5x)than test dataset.

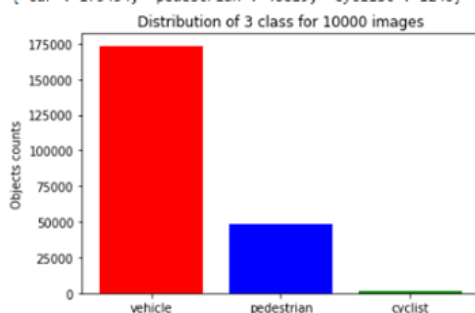
```
#display the each dataset
xlabel=["train","val","test"]
plt.bar(xlabel,[train_dataset,val_dataset,test_dataset],color=["red","green","blue"])
plt.title("Train, Validation and Test tfrecords distribution ")
```

Text(0.5, 1.0, 'Train, Validation and Test tfrecords distribution ')



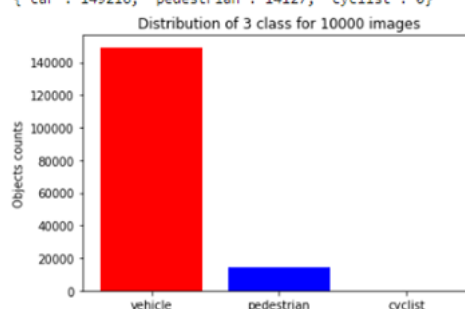
```
class_count(trainset)
```

{'car': 173434, 'pedestrian': 48819, 'cyclist': 1246}



```
class_count(testset)
```

{'car': 149216, 'pedestrian': 14127, 'cyclist': 0}



The second step is Explore augmentations.

The result is looks similar with explore the dataset.ipynb. but, this file is focused on the understanding of augmentation(manipulated the dataset to increase the dataset).

In this step, Explore the data augmentation (to increase the dataset) to understanding what is data Augmentation.

I just put some of option on the display_batch function.

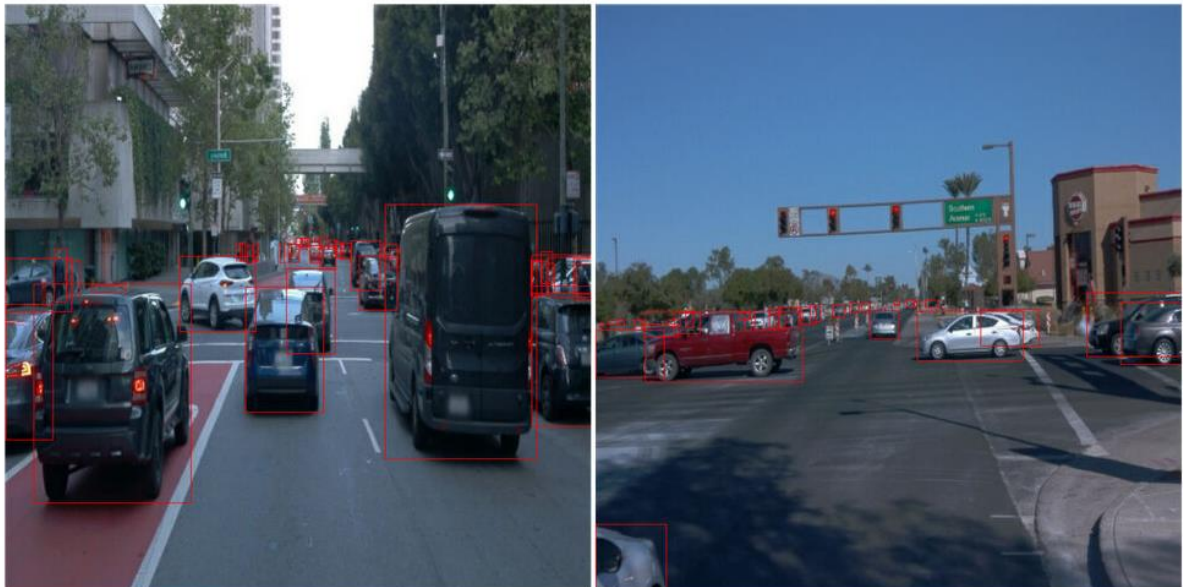
```
#plt.subplots() --> control image row and column
```

```
fig, axes = plt.subplots(1, 2, figsize=(40, 30))
```

```
axes = axes.flatten()
```

To show the images with 1 row and 2 columns intentionally.

```
%matplotlib inline  
for batch in train_dataset.take(1):  
    display_batch(batch)
```



I thought the augmentation code is included the file below

```
train_dataset = get_train_input("./experiments/reference/pipeline_new.config")
```

'pipeline_new.config'

The above file has those kind of sections below:

```

train_config {
  batch_size: 2
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
}

```

These code indicates horizontal flip to increase the dataset.

So, My guess was right! This course made a clue to find it and solve it by student by themselves!

Training

Experiments_00(iteration_00) / pipeline_new_00.config

#1. Basic setting

#2. Set basic learning rate in train_config item

#3. No settings related to adam_optimizer

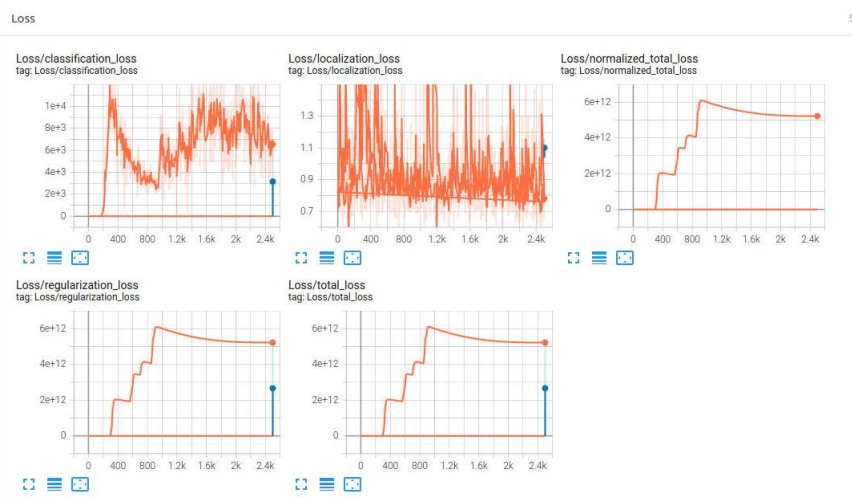
#4. manual_step_learning_rate Deprecated

```

python      experiments/model_main_tf2.py      --model_dir=experiments/reference/      --
pipeline_config_path=experiments/reference/pipeline_new.config

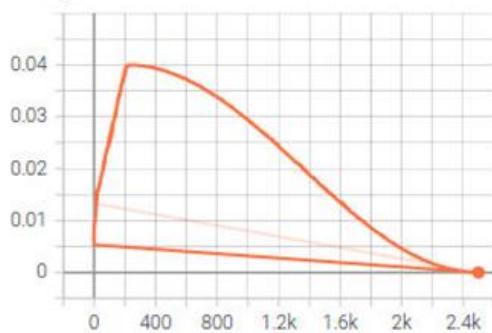
```

with this command, the result is below.



learning_rate

learning_rate
tag: learning_rate



The results shows, learning rate tends to decline. It is understandable.

And loss is tends to climb up to the top! These indicate something wrong in this anlysis.

Experiments_01(iteration_01)

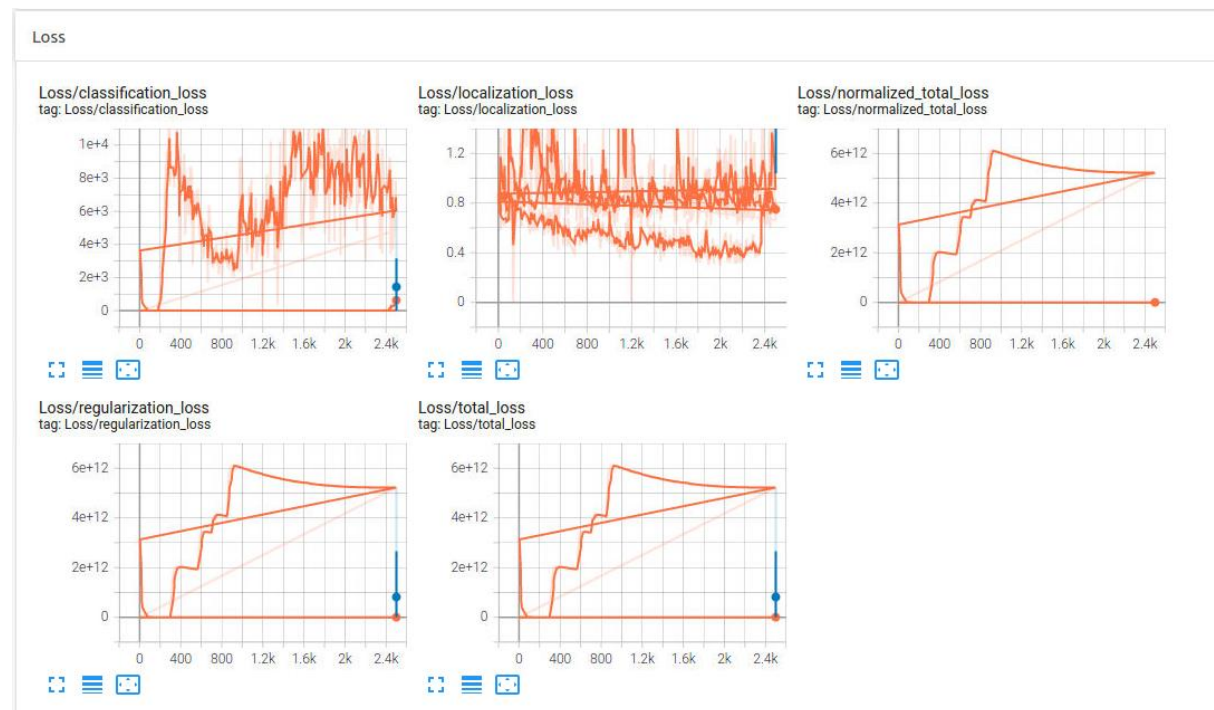
I tuned up the train_config parameters like below.

```

sync_replicas: true
optimizer {
  momentum_optimizer {
    learning_rate {
      cosine_decay_learning_rate {
        learning_rate_base: 0.04
        total_steps: 25000
        warmup_learning_rate: 0.013333
        warmup_steps: 2000
      }
    }
  }
}

```

Increasing total steps to 25000 and warmup steps to 2000





The learning rate, But results looks very not normal.

I'm considering this kind of result is not clearly reset the previous iteration result.

(home/workspace/experiments/reference/train) → log file

so, I cleaned up the log files from the location.

Experiments_02(iteration_02)

Improvements :

#adam_optimizer introduced

#1. Set adam_optimizer as learning optimization method

#2. set use_moving_average to false

What's improved

#1. More detailed learning rate adjustment simultaneously promotes rapid convergence and stable optimization in the early stages of learning.

#2. More efficient optimization possible using adam_optimizer

I tuned up the parameters like below.

```
train_config {  
  
  batch_size: 8  
  
  data_augmentation_options {  
  
    random_horizontal_flip {  
  
    }  
  
  }  
  
  data_augmentation_options {  
  
    ssd_random_crop {  
  
    }  
  
  }  
  
  data_augmentation_options {  
  
    random_adjust_contrast {  
  
    }  
  
  }  
  
  data_augmentation_options {  
  
    random_black_patches {  
  
      probability: 0.1  
  
      size_to_image_ratio: 0.05  
  
    }  
  
  }  
  
  optimizer {  
  
    adam_optimizer {  
  
      learning_rate {  
  
        manual_step_learning_rate {
```



```
    initial_learning_rate: 0.0005

    schedule {

      step: 1000

      learning_rate: 0.0001

    }

    schedule {

      step: 3000

      learning_rate: 1e-05

    }

  }

  use_moving_average: false

}

fine_tune_checkpoint: "/home/workspace/experiments/pretrained_model/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckpt-0" fine_tune_checkpoint_type: "detection"

fine_tune_checkpoint_version: V2

use_bfloat16: false

num_steps: 50000

startup_delay_steps: 0.0

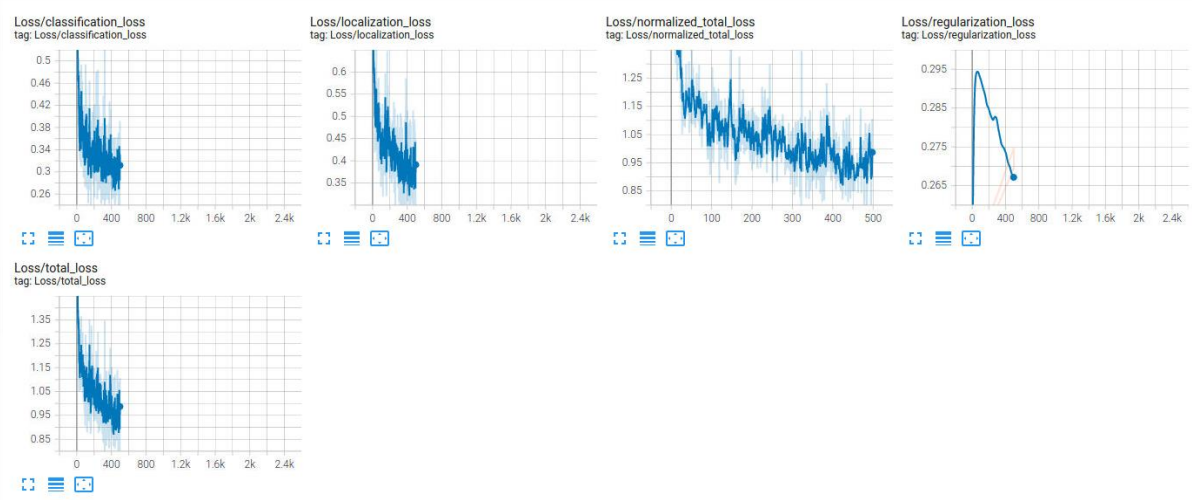
replicas_to_aggregate: 8

max_number_of_boxes: 100

unpad_groundtruth_tensors: false

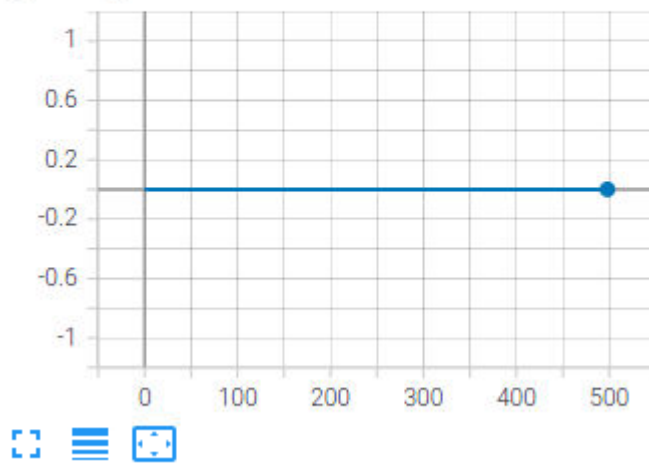
}
```

Loss



learning_rate

learning_rate
tag: learning_rate



The loss graph is understandable, but learning rate required to be fine tuned.

Experiments_03(iteration_03)

```
train_config {  
  
  batch_size: 8  
  
  data_augmentation_options {  
  
    random_horizontal_flip {  
  
    }  
  
  }  
  
  data_augmentation_options {  
  
    ssd_random_crop {  
  
    }  
  
  }  
  
  data_augmentation_options {  
  
    random_adjust_contrast {  
  
    }  
  
  }  
  
  data_augmentation_options {  
  
    random_black_patches {  
  
      probability: 0.1  
  
      size_to_image_ratio: 0.05  
  
    }  
  
  }  
  
  sync_replicas: true  
  
  optimizer {
```

```
adam_optimizer {  
  learning_rate {  
    manual_step_learning_rate {  
      initial_learning_rate: 0.0005  
  
      schedule {  
        step: 1000  
  
        learning_rate: 0.0001  
      }  
  
      schedule {  
        step: 3000  
  
        learning_rate: 1e-05  
      }  
    }  
  }  
  
  use_moving_average: false  
}  
  
fine_tune_checkpoint:  
"/home/workspace/experiments/pretrained_model/ssd_resnet50_v1_fpn_640x640_coco17_tpu-  
8/checkpoint/ckpt-0"  
  
num_steps: 5000  
  
startup_delay_steps: 0.0  
  
replicas_to_aggregate: 8  
  
max_number_of_boxes: 100
```

```

unpad_groundtruth_tensors: false

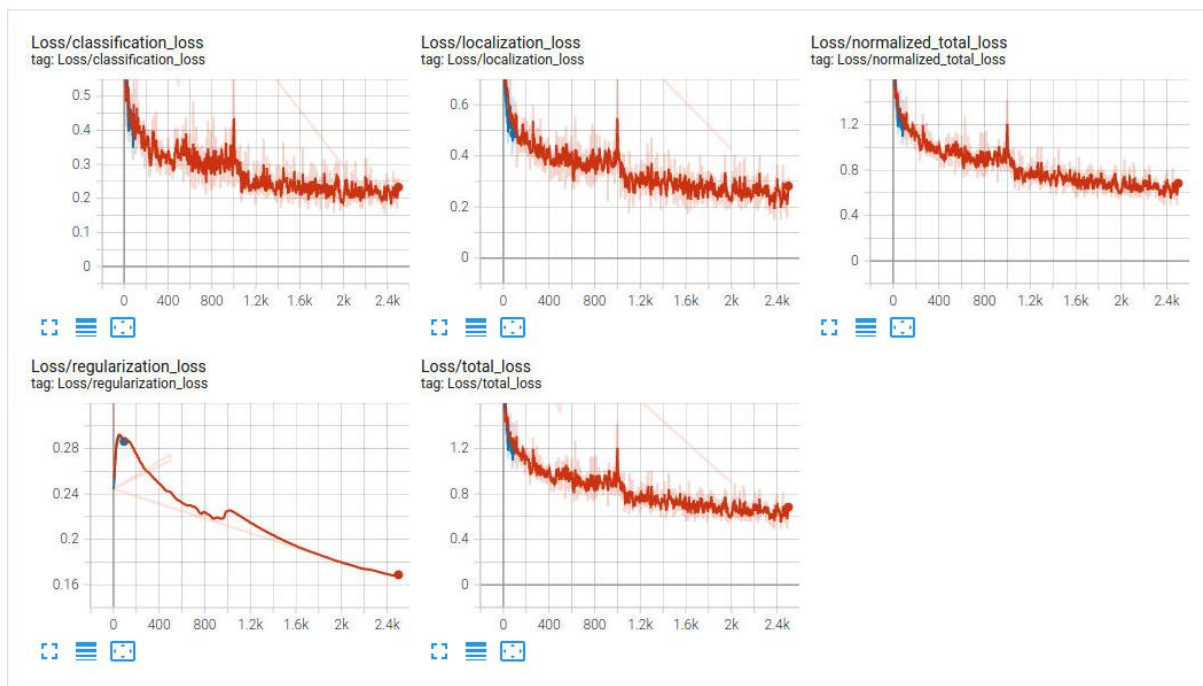
fine_tune_checkpoint_type: "detection"

use_bfloat16: false

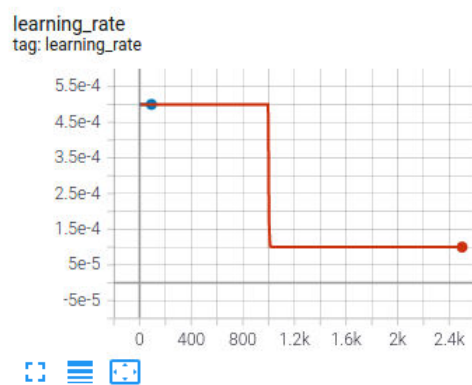
fine_tune_checkpoint_version: V2
}

```

Loss



learning_rate



The results looks pretty good, but learning rate looks strange.

It was tuned at 1000 steps and 3000steps only.

So, Finally, I updated parameter for the the next trial.

There are also experiments 04 and 05, but I will skip them.

Experiments 06(iteration 06) / pipeline_new_06.config

This is the last trial for my project.

The code is below

```
train_config {  
  
    batch_size: 8  
  
    data_augmentation_options {  
  
        random_horizontal_flip {  
  
        }  
  
    }  
  
    data_augmentation_options {  
  
        ssd_random_crop {  
  
        }  
  
    }  
  
    data_augmentation_options {  
  
        random_adjust_contrast {  
  
        }  
  
    }  
  
}
```

```
}
```

```
data_augmentation_options {
```

```
  random_black_patches {
```

```
    probability: 0.1
```

```
    size_to_image_ratio: 0.05
```

```
  }
```

```
}
```

```
sync_replicas: true
```

```
optimizer {
```

```
  adam_optimizer {
```

```
    learning_rate {
```

```
      manual_step_learning_rate {
```

```
        initial_learning_rate: 0.0005
```

```
        schedule {
```

```
          step: 1000
```

```
          learning_rate: 0.0004
```

```
        }
```

```
        schedule {
```

```
          step: 2000
```

```
          learning_rate: 0.0003
```

```
        }
```

```
        schedule {
```

```
          step: 3000
```

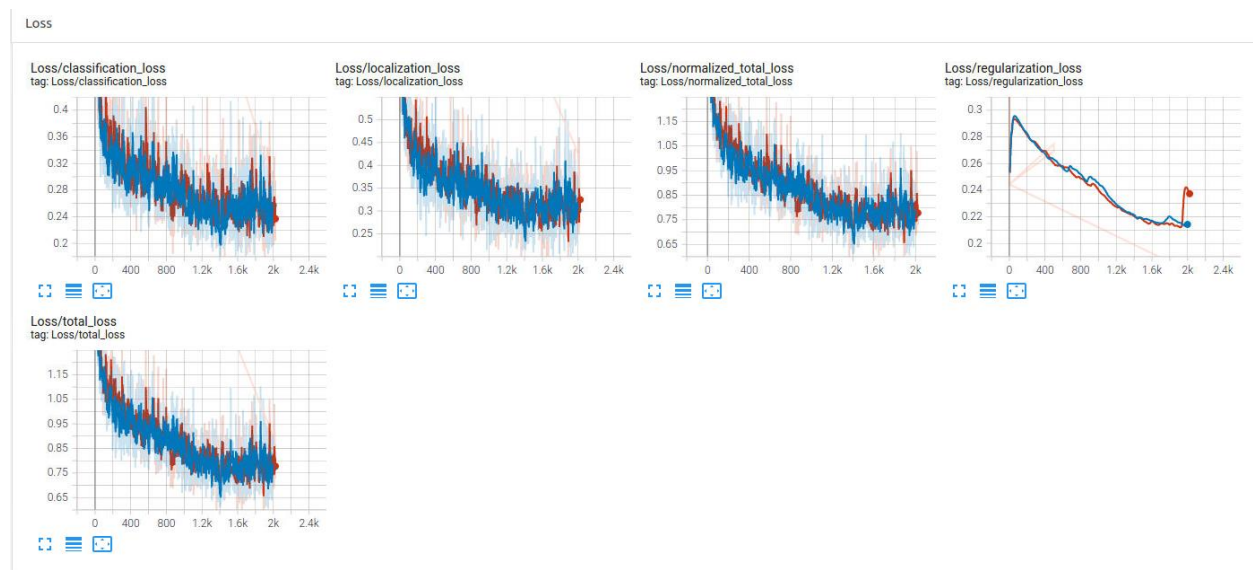
```
learning_rate: 0.0002
```

```
}
```

```
schedule {
```

```
step: 4000
```

```
learning_rate: 0.0001
```



The declined loss is reasonable. And it means analysis is going well.

I tuned the learning rate from 0.005~0.001. it is 5 stages for entire steps.

Finally result is above image.

This 'step decay' is preventing overfitting and enhancing the general performances.

#Add learning rate adjustment

#1. Added manual_step_learning_rate: Use a high learning rate for fast convergence at the beginning of training, then gradually reduce the learning rate to perform more sophisticated optimization.

#2. Initial learning rate: 0.0005, then decreasing step by step (0.0004 at 1000 steps, 0.0003 at 2000 steps, 0.0002 at 3000 steps, 0.0001 at 4000 steps)

Experiments 07(iteration 07) / pipeline_new.config

Finally, I've updates the parameters like below which given it to me from a mentor.

The code is enhanced about data augmentations.

```
train_config {  
  
    batch_size: 2  
  
    data_augmentation_options {  
  
        random_crop_image {  
  
            min_object_covered: 0.0  
  
            min_aspect_ratio: 0.75  
  
            max_aspect_ratio: 3.0  
  
            min_area: 0.75  
  
            max_area: 1.0  
  
            overlap_thresh: 0.0
```

```
    }  
}  
data_augmentation_options {  
  random_horizontal_flip {  
    keypoint_flip_permutation: 1  
    keypoint_flip_permutation: 0  
    keypoint_flip_permutation: 2  
    keypoint_flip_permutation: 3  
    keypoint_flip_permutation: 5  
    keypoint_flip_permutation: 4  
    probability: 0.5  
  }  
}
```

```
data_augmentation_options {  
  random_rotation90 {  
    keypoint_rot_permutation: 3  
    keypoint_rot_permutation: 0  
    keypoint_rot_permutation: 1  
    keypoint_rot_permutation: 2  
    probability: 0.1  
  }  
}
```

```
data_augmentation_options {
```

```
random_black_patches {  
    max_black_patches: 5  
    probability: 0.25  
    size_to_image_ratio: 0.06  
}
```

```
}
```

```
data_augmentation_options {  
    random_adjust_brightness {  
        max_delta: 0.1  
    }  
}
```

```
}
```

```
data_augmentation_options {  
    random_adjust_contrast {  
        min_delta: 0.8  
        max_delta: 1.0  
    }  
}
```

```
}
```

```
data_augmentation_options {  
    random_adjust_hue {  
        max_delta: 0.01  
    }  
}
```

```
}
```

```
data_augmentation_options {
```

```
random_adjust_saturation {  
  
    min_delta: 0.85  
  
    max_delta: 1.05  
  
}  
  
}  
  
data_augmentation_options {  
  
    random_distort_color {  
  
        color_ordering: 1  
  
    }  
  
  
  
  
  
  
  
  
  
}  
  
sync_replicas: true  
  
optimizer {  
  
    adam_optimizer {  
  
        learning_rate {  
  
            manual_step_learning_rate {  
  
                initial_learning_rate: 0.0005  
  
                schedule {  
  
                    step: 1000  
  
                    learning_rate: 0.0004  
  
                }  
  
            }  
  
            schedule {
```

step: 2000

learning_rate: 0.0003

}

schedule {

step: 3000

learning_rate: 0.0002

}

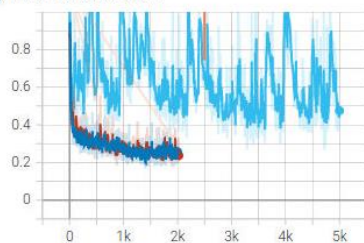
schedule {

step: 4000

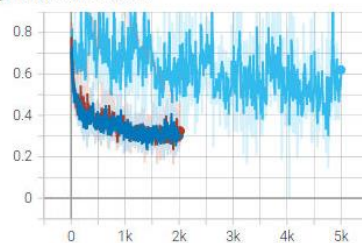
learning_rate: 0.0001

Loss

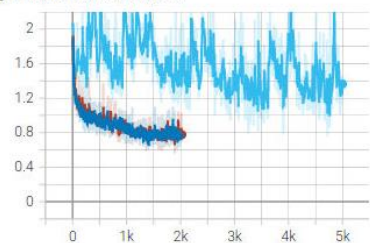
Loss/classification_loss
tag: Loss/classification_loss



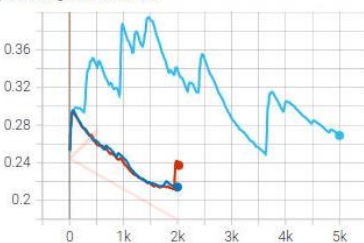
Loss/localization_loss
tag: Loss/localization_loss



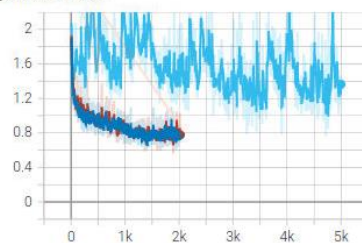
Loss/normalized_total_loss
tag: Loss/normalized_total_loss

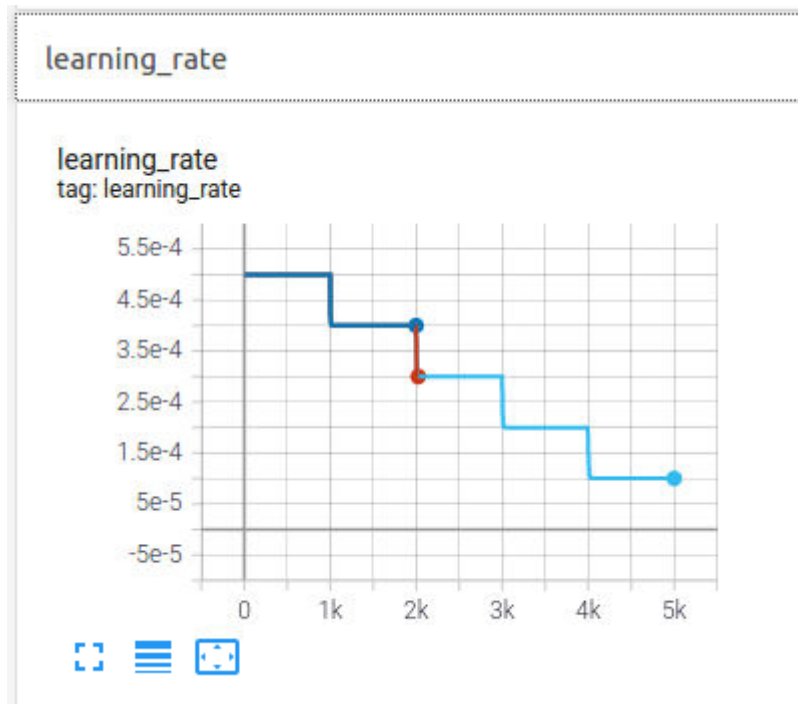


Loss/regularization_loss
tag: Loss/regularization_loss



Loss/total_loss
tag: Loss/total_loss





I Added a data augmentation parameters which given from a mentor.

It shows a flipped and grayscale and tilted images to enhance dataset.

And learning rate is gradually declined which reflect the learning rate parameters!

(iteration 06 has same learning rate parameters but it is not shown the result like this)

Conclusion (for Learning rate)

In our training process, I used a step-wise learning rate schedule. This means we started with a higher learning rate to help the model learn quickly at the beginning. As training continued, we lowered the learning rate to fine-tune the model and prevent it from making big changes that could miss the best settings.

Research supports this approach. The paper "Learning an Adaptive Learning Rate Schedule" shows that adjusting the learning rate during training can improve results significantly. By following this method, I saw the training loss go down gradually, leading to better and more stable training.

Compared learning rate between Iteration 6 and 7.

The Learning rate parameter is same but on the iteration 7 learning rate is gradually declined.

Although, Iteration 6 and 7 has same Learning rate parameters but iteration 6 has now shown the vertically declined learning rate. It shown a vertical dropping from 1000steps.

But iteration 7 has shown the exact result which reflect the learning rate parameters what intend to.

It means the data augmentation affects on the learning rate!!

References:

Learning an Adaptive Learning Rate Schedule

<https://ar5iv.labs.arxiv.org/html/1909.09712>

<https://arxiv.org/abs/1909.09712>