Udacity Self-Driving Car Engineer

Object Detection Project

Jason Twedt

11/21/2022

Project Overview

Autonomous vehicles rely on automatic perception of its surroundings in order to safely navigate the roadways. A critical component within the perception system is automatic object detection. Object detection networks rapidly processes raw sensor input to detect and classify objects surrounding the vehicle. This data is used by the vehicle's navigation system for control decisions. In this project, the Waymo open dataset and the TensorFlow Object Detection API is used to build a 2D object detection model for localization and classification of three unique object types within a self-driving car's camera field of view.

Setup

The project was setup to run locally. Instructions on how to replicate the development environment can be found in the code repository.

Dataset

Dataset overview

This project uses the *waymo_open_dataset_v_1_2_0_individual_files* bucket. 100 files representing camera video, captured at 10 frames per second (fps), with annotations for 2-dimensional object detection of three categories, listed in Table 1.

Table 1: Class Categories

Class Id	Class Category
1	Vehicle
2	Pedestrian
4	Cyclist

All 100 training, testing, and validation, source files were down-sampled to 1fps to minimize similar image examples during training.

Dataset analysis

The extracted data includes image samples of vehicles taken in both congested and non-congested regions while in a variety of environmental conditions, e.g. rain, fog, sunny, and clear. An example image with annotated bounding box overlays is shown in Figure 1. The source files illustrate a wide variety of number of objects present per frame, ranging from less than10 to greater than 60. Refer to Figure 2.

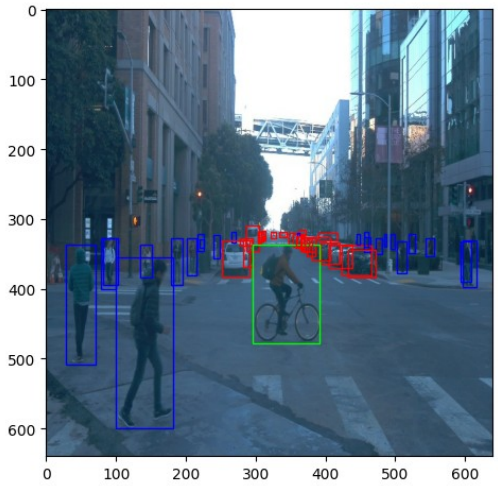

Figure 1: Sample image with annotated bounding boxes. Red: Vehicle, Blue: Pedestrian, Green: Cyclist

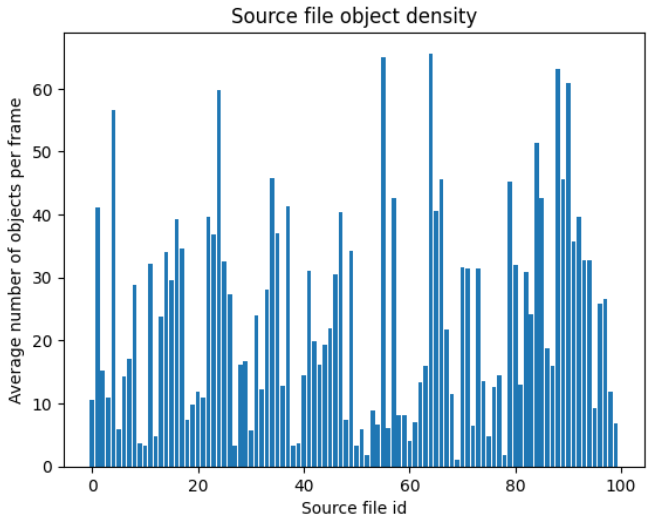

Figure 2: Average Object Density for Dataset

The estimated class distribution of the source files is roughly 76.6% vehicle, 22.8% pedestrian, and 0.6% cyclist (taken from a random sample of 10000). Refer to Figure 3.

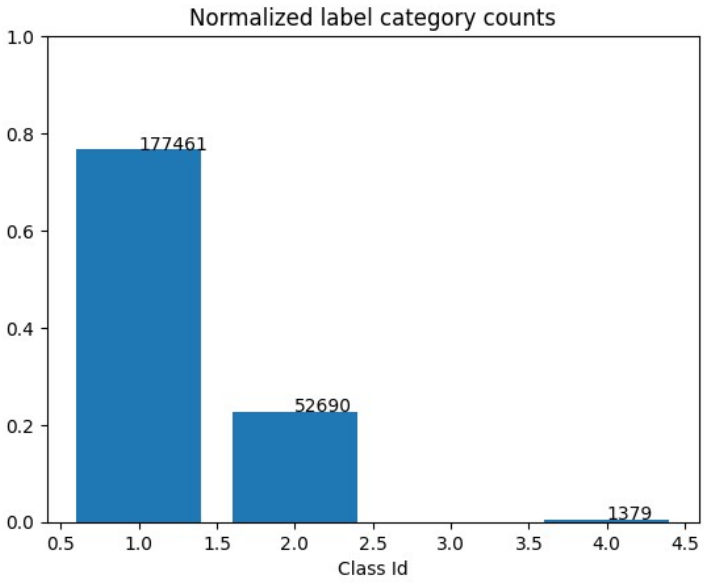

Figure 3: Class Distribution for Dataset

The class density per frame, Figure 4, indicates that cyclist most often occur as a single occurrence within a frame and are not present in groups of greater than 10 per frame. Pedestrians predominantly occur in groups of 4 or less per frame with less frequent occurrences of groupings 5 to 40 per frame. Vehicle occurrences are nearly uniformly distributed across groups of 1 to 30 per frame. A small proportion of frames have dense groupings of vehicles, 30 to 65.

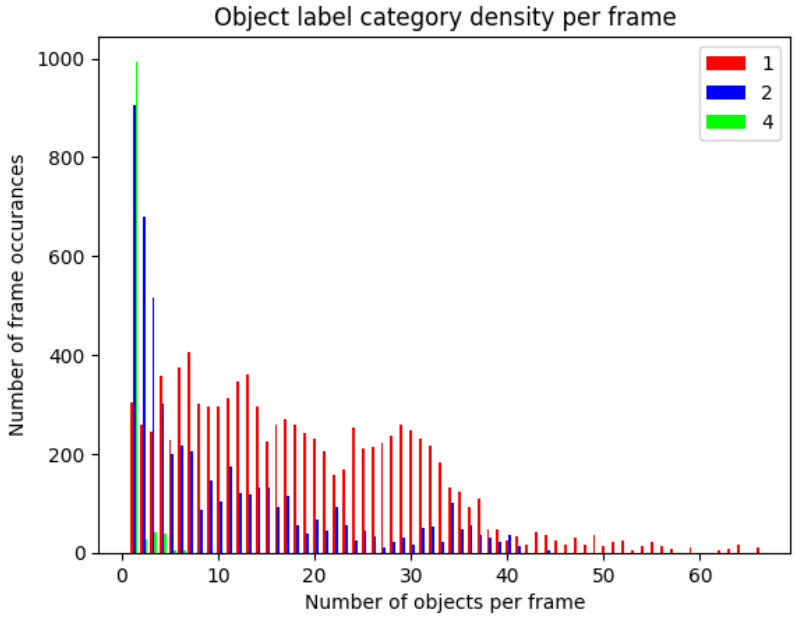

Figure 4: Object Category Density Per Frame

Cross validation

A stratification approach was used to ensure that each data split (train, test, validation) contained a proportional distribution of objects. This was implemented using the sklearn train_test_split function for multi-class classification, where object density is considered a label, represented as a one-hot vector of density bins. The dataset was split into 85% train, 9% validation, and 6% test with a maximum of 100 objects per frame and 5 total density bins. The resulting splits maintain proportional object density across all three splits, refer to Figure 5.

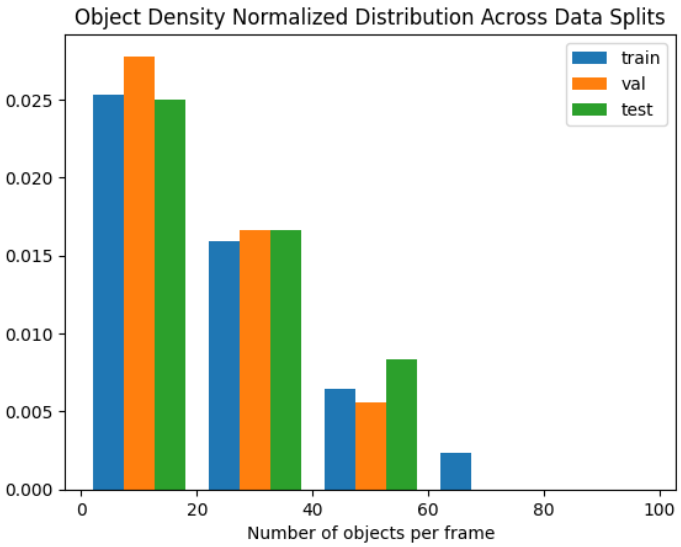

Figure 5: Object Density Distribution Across Data Splits

Training

Reference experiment

The TensorBoard output for the reference experiment is shown in Figure 6. The total loss, after an initial spike, monotonically decreases over 25k steps, indicating that the model is learning.

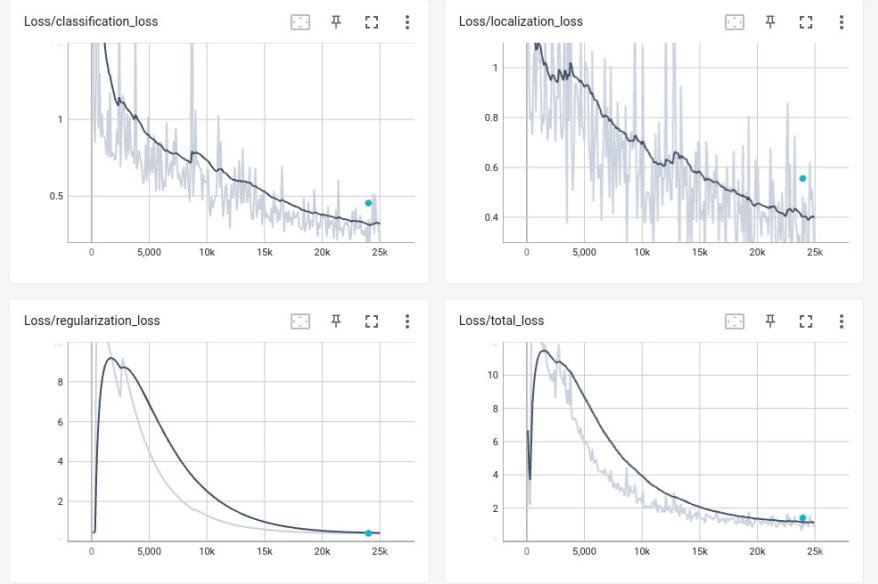

Figure 6: Reference Training Loss

The model was evaluated using the evaluation dataset. COCO evaluation metrics from are summarized below:

- Average Precision (AP) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.044
- Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.108

Very low scores for precision and recall indicate a high rate of false positive and false negatives. A sample image from the evaluation, Figure 7, visually displays the models inability to detect distant objects as well as occluded objects.

Figure 7: Reference model side-by-side evaluation

Improve on the reference

One approach to improving the reference model is to add additional augmentations during training. Image augmentations artificially simulate different environments and can increase inference accuracy on unseen examples. This dataset captures data from a variety of conditions, including: rain, fog, glare, truncated objects, obscured objects, and small objects. To increase variation of these conditions, the following augmentations were added (example outputs in Figure 8):

- random_image_scale: Enhance small to large object detection.
- random_horizontal_flip: Enhance scene variety.
- random_distort_color, random_jpeg_quality: Enhance robustness to blur and color distortions.
- ssd_random_crop_fixed_aspect_ratio: Enhance local object detection.

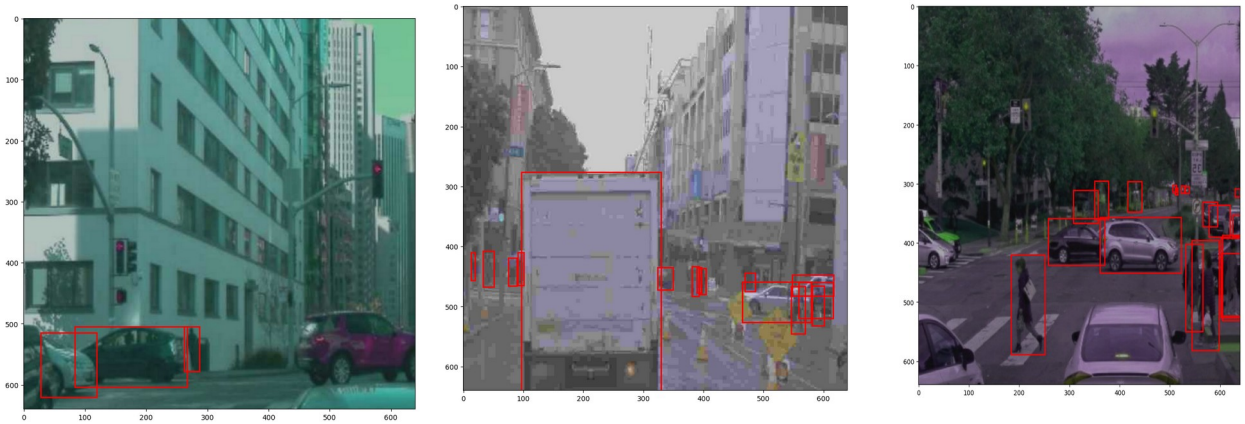

Figure 8: Image augmentation examples.

Additionally, the number of training steps was increased from 25k to 100k. This model was evaluated in a similar way to the reference with the following results:

- Average Precision (AP) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.085
- Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.152

Modest improvement was observed with this change. Another approach is to consider a new architecture. In a separate training experiment, CenterNet HourGlass 104 512x512 was used and yielded the following COCO evaluation results after 100k training steps:

- Average Precision (AP) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.158
- Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.256

Local hardware constraints prevented training these model architectures to convergence. Given more training time and imbalanced dataset corrections, e.g., oversampling minority class, both model variants would improve and potentially be suitable for real-time deployment, Figure 9.

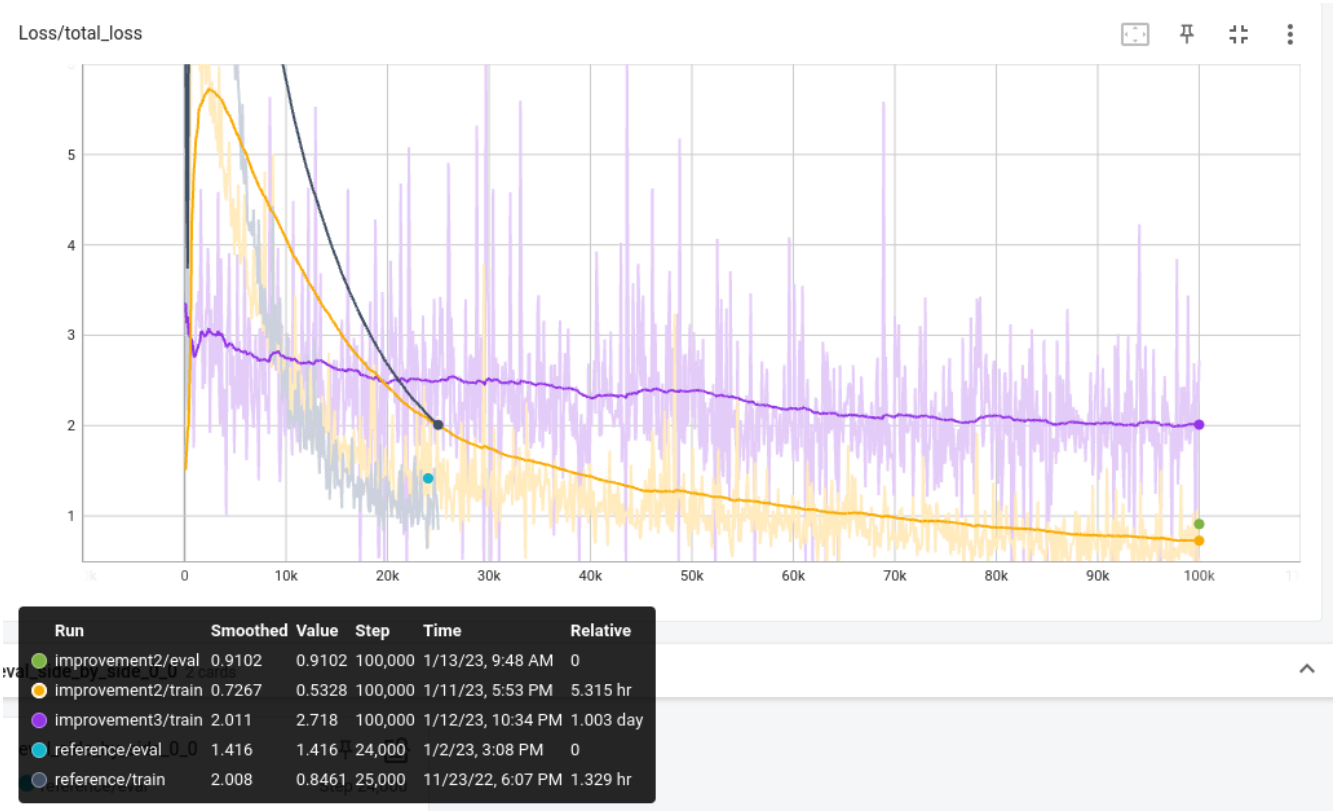

Figure 9: Training loss for all experiments.