San Diego State University

EE798 project

Real-time vehicle and pedestrian object detection and classification on the Coral EdgeTPU Board for Surrogate Safety Measurements

Written by:
Arya Yazdani

Supervised by:
Christopher Paolini

SDSU IoTLab
Department of Electrical and Computer Engineering

Summer semester 2021

August 10, 2021

San Diego State University
Department of Electrical and Computer Engineering
http://electrical.sdsu.edu

Title:
Real-time vehicle and pedestrian object detection and classification on the Coral EdgeTPU Board for Surrogate Safety Measurements

Theme:
On-device (Edge) Deep Learning

Project period:
Summer semester 2021

Project group:
SDSU IoTLab

Participant:
Arya Yazdani

Supervisor:
Christopher Paolini

Number of pages: 36

Date of completion:
August 10, 2021

Abstract

The goal of this project is to use real-time computer vision methods to detect vehicles, pedestrians and skateboarders to perform automated Surrogate Safety Measurements. More skateboarders, scooter users, and bicycle riders are using sidewalks and streets and this use of shared space increases chances of hazardous encounters between vehicles and pedestrians. This work is of significance due to the fact that skateboarders, scooter riders, and bicyclists are more prone to fatal accidents with cars since they are travelling at higher speeds and might change their direction of movement in a short amount of time, compared to pedestrians. We have implemented an on-device deep-learning architecture that performs in situ vehicle detection, and classification; and can be used for real-time vehicle tracking to compute Time-To-Collision (TTC) and Post Encroachment Time (PET) Surrogate Safety Measurements (SSM). Time-To-Collision is a Surrogate Safety Measure that estimates an expected collision time for two or more vehicles in motion. Post-encroachment time is the time interval between two instances when one vehicle leaves a conflict point (or zone area) and a second vehicle enters into the same conflict point (or zone area). An outcome of this work is a model that can be deployed on low-cost, small-footprint mobile and IoT devices at traffic intersections with existing cameras to perform on-device inferencing

for in situ Surrogate Safety Measurement (SSM), such as Time-To-Collision (TTC) and Post Encroachment Time (PET). SSM values that exceed a hazard threshold can be published to an Message Queuing Telemetry Transport (MQTT) broker, where messages are received by an intersection traffic signal controller for real-time signal adjustment, thus contributing to state-of-the-art vehicle and pedestrian safety improvement at hazard-prone intersections. Surrogate Safety measures can be evaluated using deep learning models, such as Faster R-CNN and SSD (Single-Shot Multibox Detector). In a previous work, detection and classification of pedestrians and skateboarders, and computation of SSMs, has been done using captured images of different camera perspectives of a three-way traffic intersection, and Faster R-CNN and SSD Multibox models have been trained offline to detect and classify pedestrians and skateboarders. An outcome of this effort is a small form factor electronic device, enclosed in a water-tight case, and mountable on traffic light beams, which computes and transmits real-time TTC and PET measurements for use in traffic control systems. A prototype will be developed at SDSU and installed on campus next to our existing Pelco Esprit® traffic camera located on the 6th story balcony of the GMCS building, which monitors a three-way intersection on campus. Safety measurements and video will be streamed back to a nine-screen video wall at SDSU for evaluation. This project involves on-device machine learning where TensorFlow is used to train the MobileNetV2 model with images of vehicles, pedestrians, bicycles, and skateboards at an intersection being monitored on campus, and then a Coral EdgeTPU device is programmed using TensorFlowLite with the EdgeTPU-optimized counterpart MobileNetEdgeTPU model to perform continuous inferencing on a live Real Time Streaming Protocol (RTSP) video stream.

Table of Contents

1 Introduction

With increased use of skateboards, scooters, and bicycles as means of urban transportation, especially for short-distance itineraries, the probability of collision between automobiles and other road users is rising. The popularity and affordability of personal electric skateboards makes these devices a good candidate as a near future dominant transportation tool for short distance travel [1]. In recent years, electric bicycles (e-bikes) and electric scooters (e-scooters) have become a popular means of transportation for short itineraries and so-called "last mile" travel options in cities and metropolitan areas. These devices can increase the chances of collision and create a safety hazard between each other, pedestrians, and other vehicles [2]. Pedestrians on sidewalks are vulnerable too, and must avoid rapidly moving skateboarders. Therefore, foreseeing the implementation of further safety measures and infrastructure does not seem unrealistic. One mechanism for estimating regions of a thoroughfare shared with multiple vehicle types are known as Surrogate Safety Measures (SSMs). These measurements provide a probability of near-collision events by measuring spatial and temporal proximity between road users. Two SSM parameters of interest are Time-To-Collision (TTC) and Post Encroachment Time (PET). Time-To-Collision estimates an expected collision time for two or more vehicles in motion. Post-encroachment time is the time interval between two instances when one vehicle leaves a conflict point (or zone area) and a second vehicle enters into the same conflict point (or zone area).

One promising technique to improve traffic safety is to use computers and cameras to implement Artificial Intelligence (AI); Taking advantage of computer vision techniques has gained popularity with advancements in deep convolutional neural networks (CNN) [3]. Image classification and object detection are two objectives for which CNNs are widely being used. Recently, deep convolutional networks have significantly improved image classification and object detection [4]

A CNN is comprised of numerous layers and nodes which can classify images fed into it and detect the bounding boxes around specific objects if those objects exist in the image. Convolutional layers and pooling are among the main elements in a CNN [5]. Single Shot Multi-box Detection (SSD) is a type of CNN which is able to detect multiple objects in an image [6]. Compared to image classification, object detection is a more challenging task that requires complex methods.

Mobilenet [7] is a relatively small-network, low latency model that can be trained and deployed to match the design requirements of mobile and embedded vision applications. MobileNets are built primarily from depth-wise separable convolutions to reduce the computation in the first few layers. In this project, Tensorflow is used to train the Mobilenet version 2 (v2) model with images that include different *objects* (e.g. vehicles, pedestrians, and skateboarders). In this report, the detailed steps that have been taken

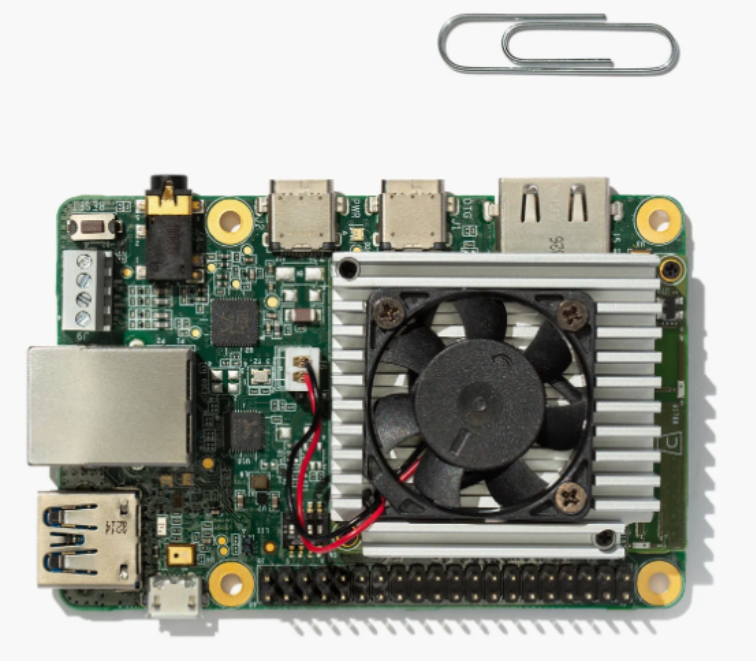


Figure 1.1: *Google Coral Dev Board*

to generate a model, which runs inferencing on a live video stream and is capable of real-time detection and classification of objects, are presented. A lightweight version of the Mobilenet model needs to be generated in order for real-time detection. Lightweight models in Tensorflow are called *TFLite* models. In this project we focus on the steps taken to generate TFLite models that can be used for object inferencing on an edge device, such as the Coral EdgeTPU board [8] shown in figure 1.1. The Dev board is a single-board computer ideal for fast machine learning inferencing. Compatibility with TFLite models makes this board a very good candidate to use models that are trained using Tensorflow. Table 1.1 shows the specs of the board [8]

1.1 Outline of report

This work is composed of different sections as follows:

Chapter 2 includes two sections; Section 2.1 includes a quick review of two different CNN's that are used in object detection; SSD (Single-Shot Multibox Detector) and FPN (Feature Pyramid Network). Later in the report, we discuss the difference among these models in terms of training, validation, and deployment. In Section 2.2, important metrics that indicate the performance of an object detection model are presented. These metrics are used to evaluate the model we used for real-time detection.

In Chapter 3, the steps taken to capture images that include desired objects on a live camera stream are discussed. We also discuss how these images were manually annotated.

Training the CNN is discussed in Chapter 4. Transfer learning (also known as *fine*

Table 1.1: *Coral Dev Board Specs [8]*

CPU	NXP i.MX 8M SoC (quad Cortex-A53, Cortex-M4F)
GPU	Integrated GC7000 Lite Graphics
ML accelerator	Google Edge TPU coprocessor: 4 TOPS (int8); 2 TOPS per watt
RAM	1 GB LPDDR4 (option for 2 GB or 4 GB coming soon)
Flash memory	8 GB eMMC, MicroSD slot
Wireless	Wi-Fi 2x2 MIMO (802.11b/g/n/ac 2.4/5GHz) and Bluetooth 4.2
USB	Type-C OTG; Type-C power; Type-A 3.0 host; Micro-B serial console
LAN	Gigabit Ethernet port
Audio	3.5mm audio jack (CTIA compliant); Digital PDM microphone (x2); 2.54mm 4-pin terminal for stereo speakers
Video	HDMI 2.0a (full size); 39-pin FFC connector for MIPI-DSI display (4-lane); 24-pin FFC connector for MIPI-CSI2 camera (4-lane)
GPIO	3.3V power rail; 40 - 255 ohms programmable impedance; ~82 mA max current
Power	5V DC (USB Type-C)
Dimensions	88 mm x 60 mm x 24mm
Availability	Australia, Japan, New Zealand, Taiwan, European Union (except France, Czech Republic), Ghana, Hong Kong, India, Oman, Philippines, Singapore, South Korea, Thailand, United States

tuning is the method used in this project to train the MobilenetV2 model. We discuss in detail how we performed transfer learning.

Chapter 5 covers the steps that were taken to export the trained model, quantization, conversion to a TFLite model, and finally compiling the model for EdgeTPU board. Examples of object detection for different object labels are also illustrated.

Discussion of the results and findings is given in Chapter 6.

Finally, Chapter 7 concludes this report and presents opportunities for future work.

2 Object Detection Models

2.1 Types of CNN Used for Training

SSD and FPN are two common CNNs used for object detection. A Feature Pyramid Network (FPN) is a feature extractor designed in a pyramid topology, as shown in Figure 2.1, optimized for accuracy and speed. FPN is a replacement for the feature extractor of detectors like Faster R-CNN and generates multiple feature map layers (multi-scale feature maps) with better quality information [9]. An FPN is comprised of a bottom-up and a top-down pathway. The bottom-up pathway is the usual convolutional network for feature extraction. The spatial resolution decreases from bottom to top. The semantic value for each layer increases as more high-level structures are detected (Figure 2.2). A SSD model makes detections from multiple feature maps. However, SSD has a downside such that the bottom layers are not selected for object detection. While being high resolution, the semantic value is not high enough to justify use, since the speed degradation is significant. Therefore, SSD only uses upper layers for detection and, as a result, performance for small object detection is not satisfactory.

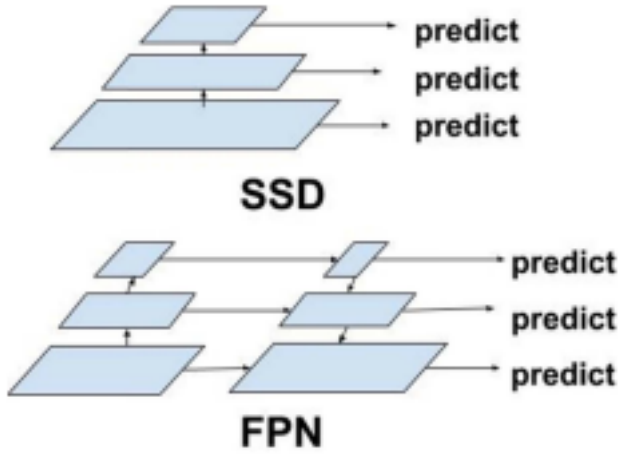


Figure 2.1: *SSD versus FPN [10]*

2.2 Performance Metrics of An Object Detection Model

To calculate the precision of an inferred object, we quickly review the concept of *Intersection over Union* (IoU). As seen in Figure 2.3, IoU is defined using the ratio in

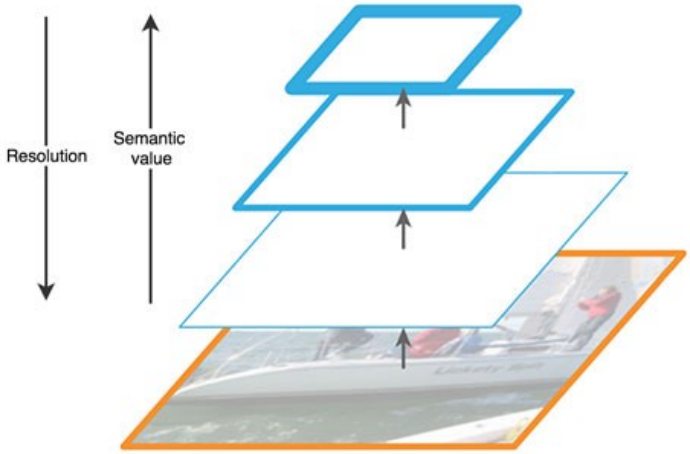


Figure 2.2: *Feature Extraction in FPN [9]*

Figure 2.4:

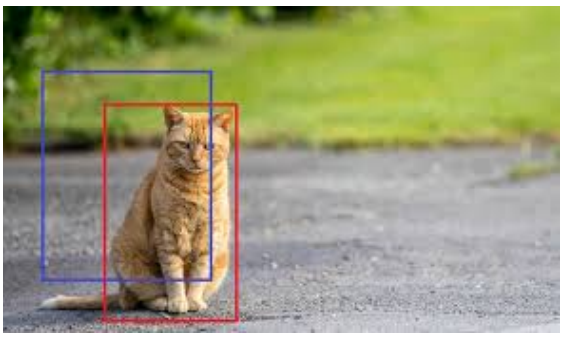


Figure 2.3: *Ground Truth in red versus detection box in blue of a cat [11]*

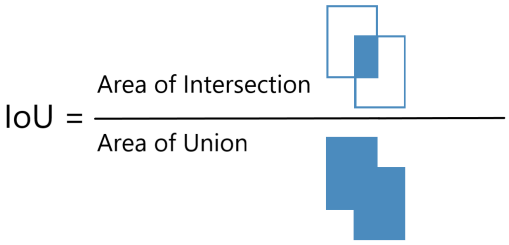


Figure 2.4: *Definition of Intersection over Union (IoU) [12]*

Mean Average Precision (mAP) is an evaluation metric used for object detection. To calculate mAP we need to evaluate precision and recall parameters, which are defined in Equations 2.1 and 2.2:

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

True Positive (TP) indicates that the network correctly predicted an object. False Positive (FP) is when the network predicts the presence of an object when there is not any. False Negative (FN) describes the condition where the network cannot predict an existing object. Average Precision (AP) is the area under the precision-Recall (PR) curve. Then mAP is obtained by averaging the AP calculated for all defined classes [13].

3 Capturing Images and Annotation

In order to obtain images for training, the Pelco Esprit® traffic camera located on the sixth story balcony of the GMCS building at San Diego State University, which monitors a three-way intersection on campus, has been used. Due to low traffic on campus in second half of year 2020 due to the COVID-19 pandemic, several videos over a course of month where recorded in mp4 format and later on, frames with desired objects including vehicles and pedestrians were selected. The following objects were of interest:

- car
- truck
- suv (sport utility vehicle)
- bicycle
- motorcycle
- van
- cart
- box-truck (e.g., a UPS delivery truck)

The Pelco camera resolution was set to 720p (1280 pixels x 720 pixels) and 5 fps (frames per second). Videos were recorded in four different pan (θ) and tilt (ϕ) angles to capture traffic using multiple perspectives. In addition, previously captured and annotated pedestrian and skateboarder images from a previous work [1] were added to the training data set. Note that there was difficulty capturing some classes like motorcycles, due to low traffic on campus. For later training and with anticipated higher traffic on campus, we hope to be able to capture more frames with motorcycles. A total of 16762 images were selected to train the model and the total number of ground truth instances for each object were 36741 and are specified in table 3.1.

The following procedure explains how we captured mp4 videos using the Pelco camera. The *notos.sdsu.edu* GPU server was primarily used to record and store all videos in addition to training the network. notos.sdsu.edu is a Supermicro SuperServer 4028GR-TR GPU cluster optimized for AI, deep learning, and/or HPC applications. notos.sdsu.edu features 8x Nvidia Tesla V100-PCIe GPUs and has the TensorFlow, TensorFlow Lite, Caffe, and Keras deep learning frameworks installed. The Pelco camera is on SDSU campus and there is a firewall that blocks access from notos.sdsu.edu to the camera. In order to access video streams from notos.sdsu.edu, a reverse ssh tunnel was established

Table 3.1: *Number of ground truth for each label used for training*

Object label	Number of images in dataset
car	1609
truck	873
suv	1191
van	774
cart	507
boxtruck	309
bicycle	1387
motorcycle	79
pedestrian	24129
skateboarder	5883

from the Pelco server to notos.sdsu.edu. This tunnel in instantiated using the following command invoked in the camera's Linux shell environment:

```
ssh -R 50000:localhost:554 user@notos.sdsu.edu
```

This command tunnels the well-known RTSP TCP port 554 on the camera, which is where the video stream is transmitted, to port 50000 on notos.sdsu.edu (an arbitrary ephemeral and available port). With the ssh command above, notos.sdsu.edu has access to Pelco live stream through local TCP port 50000. We use the ffmpeg [14] tool to record mp4 videos using the following command on notos.sdsu.edu:

```
ffmpeg -rtsp_transport tcp -i rtsp://localhost:50000/stream2 -an \
-framerate 5 -strict -2 -vcodec copy output.mp4
```

Note that command line option *-an* disables audio capture which is more efficient since the audio channel is not needed for object detection.

We recorded multiple videos over the course of several days, during both daytime and nighttime hours and at different pan and tilt angles. Captured videos were then deconstructed into frames (five frames per second) and reviewed to select final frames containing objects of interest. Extracted frames were saved in *.png* format, since PNG is a lossless bitmap image format and we want to retain maximal image detail for training. VGG Image Annotator (VIA) [15], [16] was used to annotate selected images. Rectangular bounding boxes, which included the ground truth of each object, were defined for all images. For images that included more than one object, multiple rectangular bounding boxes were drawn. The output of VGG annotator is a *.csv* file which includes rows with image name, object type, top left coordinates (x,y) of the ground truth, and the corresponding width and height of a bounding box (x,y).

A dataset that includes 10070 images of pedestrians and skateboarders [17], which was collected with the same Pelco camera and also annotated using VGG annotator, was later added to the image dataset used in this project. These images have pixel dimension of

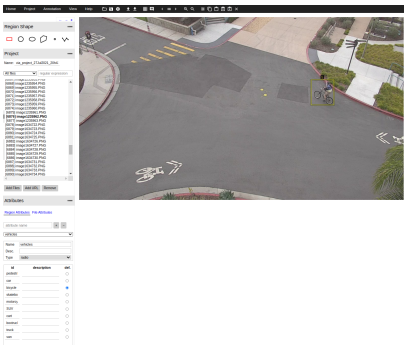
1280x720 in JPEG format (*.jpg*). In order to use these images to train MobileNetV2 in this work, all images were converted into PNG format.

(a) *bicycle* **(b)** *truck*

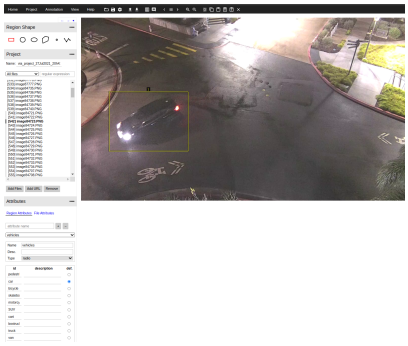
(c) *van*

Figure 3.1: *Example of images captured from our GMCS building Pelco camera and annotated using VGG with class labels bicycle, truck, and van.*

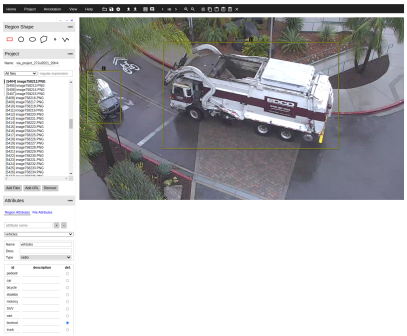
In Figure 3.2, a screenshot of using VGG annotator is shown; labels are defined in a VGG project and every image that includes an object is annotated and bounding boxes are drawn around each object with the label tag assigned. Figure 3.3 shows a snippet of comma-separated values (csv) formatted output that VGG generates. In next chapter we explain the procedure used to annotate images and train a MobileNetV2 network.



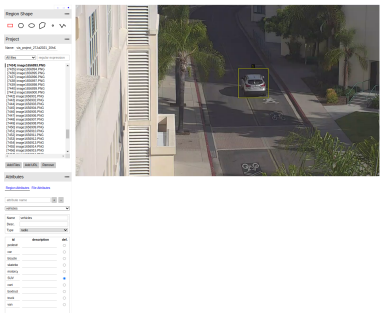
(a) *bicycle on 3-way intersection*



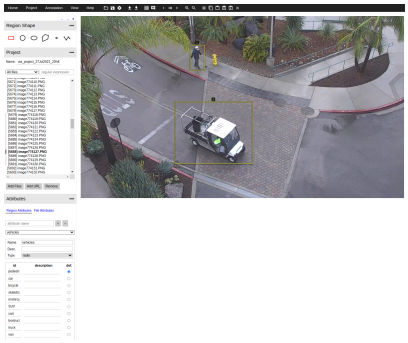
(b) *car at night-time on 3-way intersection*



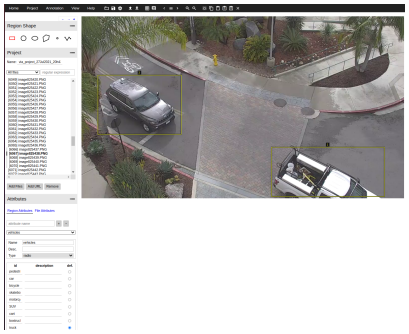
(c) *boxtruck and cart on 2-way intersection*



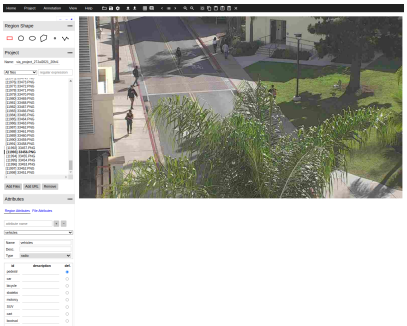
(d) *car on a left-most camera angle*



(e) *pedestrian and cart on 2-way intersection*



(f) *SUV and truck on 2-way intersection*



(g) *pedestrian and skateboarder [18],[17]*

Figure 3.2: *VGG annotator Project Environment*

	A	B	C	D	E	F	G
1	filename	xmin	ymin	xmax	ymax	vehicle_type	
2	image38595.PNG	162	71	240	184	bicycle	
3	image38596.PNG	122	53	202	178	bicycle	
4	image38597.PNG	91	45	170	148	bicycle	
5	image38598.PNG	72	18	148	143	bicycle	
6	image38599.PNG	51	16	117	121	bicycle	
7	image38600.PNG	26	0	100	105	bicycle	
8	image38601.PNG	4	3	82	85	bicycle	
9	image38602.PNG	4	4	54	70	bicycle	
10	image38586.PNG	581	83	682	208	bicycle	
11	image38587.PNG	533	113	621	220	bicycle	
12	image38588.PNG	473	112	578	225	bicycle	
13	image38589.PNG	431	119	515	219	bicycle	
14	image38590.PNG	374	108	467	229	bicycle	
15	image38591.PNG	324	124	425	219	bicycle	
16	image38592.PNG	263	99	378	215	bicycle	
17	image38593.PNG	228	101	328	208	bicycle	
18	image38594.PNG	195	76	273	200	bicycle	
19	image38577.PNG	888	12	987	83	bicycle	
20	image38578.PNG	873	9	935	91	bicycle	
21	image38579.PNG	841	20	917	115	bicycle	
22	image38580.PNG	810	33	888	126	bicycle	
23	image38581.PNG	786	41	857	141	bicycle	
24	image38582.PNG	751	55	833	160	bicycle	
25	image38583.PNG	720	62	781	187	bicycle	
26	image38584.PNG	680	74	756	175	bicycle	
27	image38585.PNG	629	80	717	195	bicycle	
28	image38571.PNG	1061	4	1124	22	bicycle	
29	image38572.PNG	1030	4	1113	34	bicycle	
30	image38573.PNG	1002	8	1089	37	bicycle	
31	image38574.PNG	980	4	1050	53	bicycle	
32	image38575.PNG	947	0	1034	62	bicycle	
33	image38576.PNG	915	3	1019	81	bicycle	
34	image96405.PNG	719	74	832	215	van	
35	image96406.PNG	723	59	818	214	van	
36	image96407.PNG	724	43	829	179	van	
37	image96408.PNG	722	29	838	167	van	
38	image96409.PNG	724	26	860	150	van	
39	image96410.PNG	740	13	851	130	van	
40	image96411.PNG	753	5	864	122	van	
41	image96412.PNG	765	4	868	109	van	
42	image96413.PNG	773	1	884	91	van	
43	image96414.PNG	766	1	891	89	van	
44	image96415.PNG	790	5	889	64	van	
45	image96395.PNG	727	221	1018	401	van	
46	image96396.PNG	722	199	971	395	van	
47	image96397.PNG	722	175	951	374	van	

Figure 3.3: *Output of VGG annotator as csv file including image name, bounding boxes and label type*

4 Training the Mobilenet Network

MobileNetV2 is an already trained classification network with 90 different labels from the COCO large-scale object detection, segmentation, and captioning dataset [19]. In this work, annotated images of vehicles and pedestrians need to be used for a fine-tuning of the pre-trained network and adjusting the classification labels. This technique is known as *transfer learning*, which preserves potentially useful features included in the initial model [20]. Additionally, an object detection feature is added to the network after training to enable the model to be able to both detect an object (report bounding box coordinates) and classify the detected object based on our specified class labels.

720p (HD or *high definition*) display resolution images of dimension 1280 x 720 pixels^2 were stored on notos.sdsu.edu. Training using TensorFlow was also performed on notos.sdsu.edu. The model used in this project is Mobilenet_v2 with an image input dimension of 300x300 pixels^2. Our captured and annoteted images 720p resolution were scaled to 300x300 pixels^2. To train Mobilenet_v2 using transfer learning, we performed the following procedure:

1. Splitting images into separate training and testing datasets

   A script was downloaded [21] and updated to split all images with a ratio of 0.9 for training and 0.1 for testing (see B. The annotation .csv file generated by VGG annotator was updated to have separate columns for xmin, xmax, ymin, ymax, and object class for each image. The script used to split the image dataset creates two separate csv files for training and testing data.

2. Fine tuning with TensorFlow

   In order to use fine tuning, TensorFlow has the capability of creating TensorFlow Record (TF) binary file. A python script was used to create TF records with image data, bounding box coordinates, and object classes. Two TF record files were generated for the training and testing datasets, which are used in a pipeline configuration file to train the network.

3. TensorFlow pipeline configuration

   An important step in transfer learning is defining parameters within a *'pipeline.config'* file. This file includes information about parameters of the model, such as the number of training steps to perform. The parameter batch size needs to be configured based on desired speed of training and operating system capability. A batch size of 128 was specified, which means at each training step, 128 images are fed to the network input layer. The following seven parameters are among the most important

ones to check and update if needed in pipeline.config which was downloaded from repository in [22];

- number of classes: 10. This value is updated to 10 because we have 10 object classes.
- fixed_shape_resizer height: 300, width: 300. This is default value. Image size input in a MobilenetV2 SSD is 300x300.
- feature extractor type: ssd_mobilenet_v2_keras. This parameter defines the type of feature extractor, either it is SSD or FPN. SSD was default parameter in this config.
- batch size: 128. The default value was 512; bigger batch size can speed up training but it might affect accuracy of the network since it directly changes stochastic gradient descent calculation [23]. Value of 128 was selected after some training efforts.
- learning_rate_base: 0.079. This is default value in the config.
- num_steps: 60000. Value updated after some training and leads to satisfactory accuracy.
- fine_tune_checkpoint_type: "detection". This parameter should match training objective; if the objective is classification only, then should be updated accordingly; classification is default in the config

4. Training

   We configured Python script *"model_main_tf2.py"*, included in the appendix, for training.

   This script is updated with the path to *'pipeline.config'*, an empty folder (named *model*) to deposit training related files and for saving a checkpoint every 1000 steps, which can be used to export the model at a particular desired step.

Training the network on Nvidia V100 GPUs took around 12 hours. With TensorFlow, we can run Tensorboard while training to monitor the process. Figure 4.1 shows a sample of images used for training at a step 23078.

Figure 4.2 illustrates the learning rate. It is seen that this is a dynamic learning rate which decays to zero at step 60k. This figure indicates that the model has converged and is well fit to the data set and further training will not improve the model. In addition, the fact that this model is well fit is seen in Figures 4.3, 4.4, and 4.5. Localization loss refers to the loss related to finding the bounding boxes. Localization loss of 0.06 at the end of training is obtained.

Classification loss is related to the training loss on the object within a bounding box. At the end of training, the amount of this loss is approximately 0.08, which shows the model is well fit to the image dataset.

Total loss includes all different losses during training. The value of 0.19 is observed at the end of training; the decay of all losses during the training is a very good sign that the model is converging and fits the dataset.

Figure 4.1: *A sample of training images that were fed into the network at step 23078.*

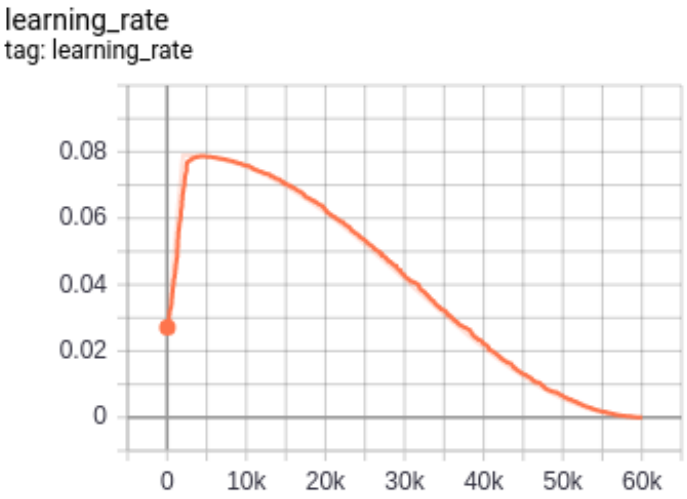


Figure 4.2: *Learning rate versus training step number.*

Figure 4.6 shows the number of steps per second (frequency). Note that in each step, a batch size of 128 images are introduced into the network. In this training, we observe an average of 1.5 steps per second.

4.1 Training the Mobilenet V2 FPN (Feature Pyramid Network)

We initially deployed a *Mobilenet_v2_fpn* model with input image dimension 640x640 pixels^2 on an EdgeTPU device. Feature Pyramid Network models are briefly explained in

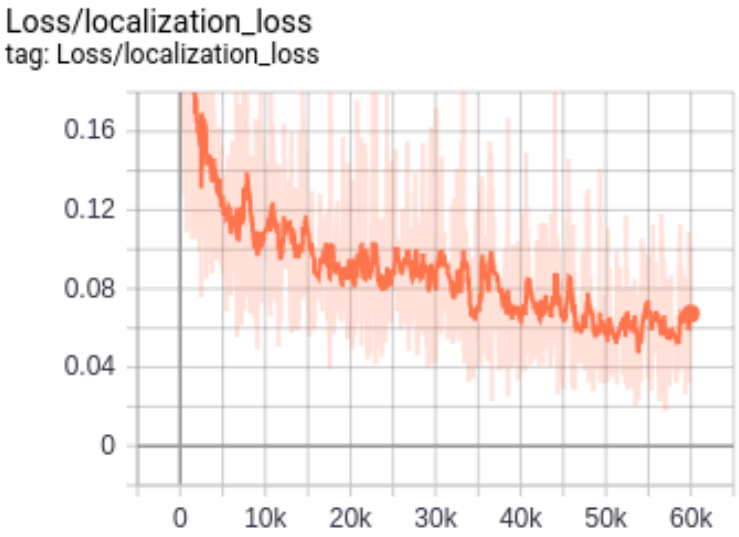


Figure 4.3: *Localization loss during training.*

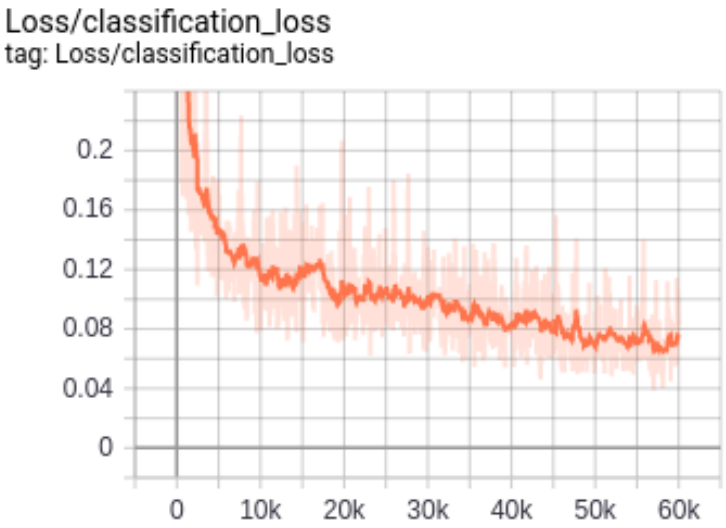


Figure 4.4: *Classification loss during training.*

Section 2.1. We found the FPN 640x640 pixels^2 model resulted in poor inference times on the EdgeTPU device, unacceptable for real-time deployment in a production traffic intersection setting. Therefore, we elected to deploy the SSD model *Mobilenet_v2_300x300* on the EdgeTPU device, while reporting a summary of the FPN model performance in Section 6.

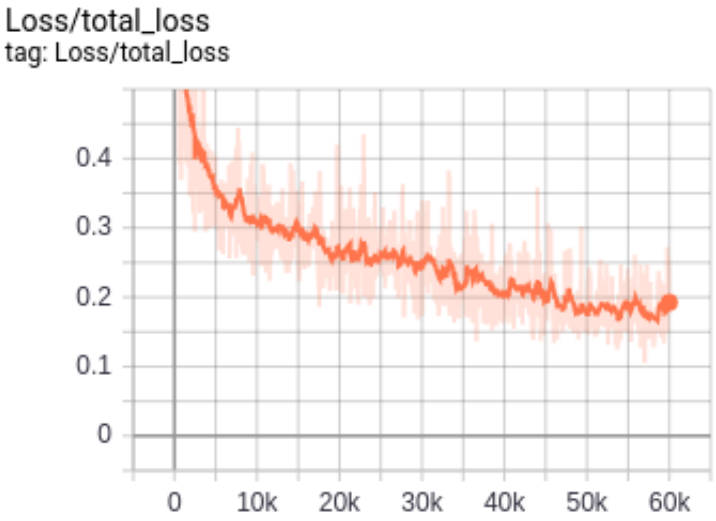


Figure 4.5: *Total loss during training.*

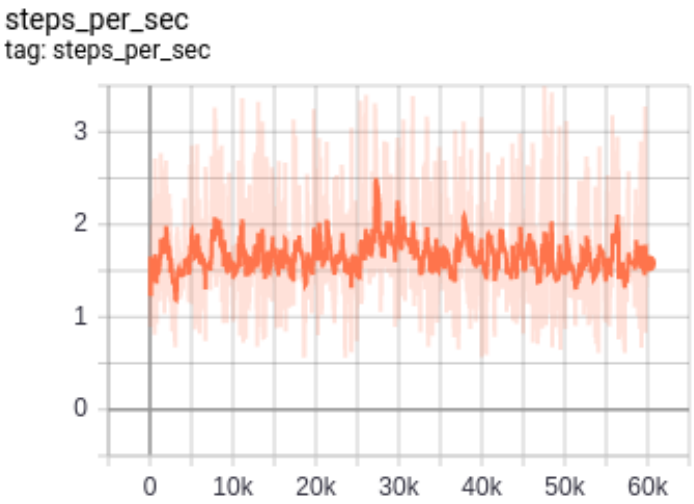


Figure 4.6: *Step frequency during training.*

5 Exporting and Conversion to a TFLite model

In this section we discuss exporting (a.k.a. *freezing*) a trained model, quantization, conversion to TFLite, and compiling the model to be used on an EdgeTPU board.

5.1 Freezing a trained model

After training has completed, and results in terms of total loss are found to be satisfactory, we export the model to a frozen graph. At step 60k in training, total loss reached a plateau at a value of 0.2 (as seen in Figure 4.5). since we had checkpoints generated at every 1000 steps, we can use the last generated checkpoint, which is for step 60k, to freeze the model. The steps used to export a model depend on the network application. Script *'exporter_main_v2.py'* exports a model using a checkpoint and *'pipeline.config'* file, where the exported model is suitable for use on a CPU, with slower inference time. This exported model is not intended for conversion to TFLite. The command to export the model using *'exporter_main_v2.py'* is

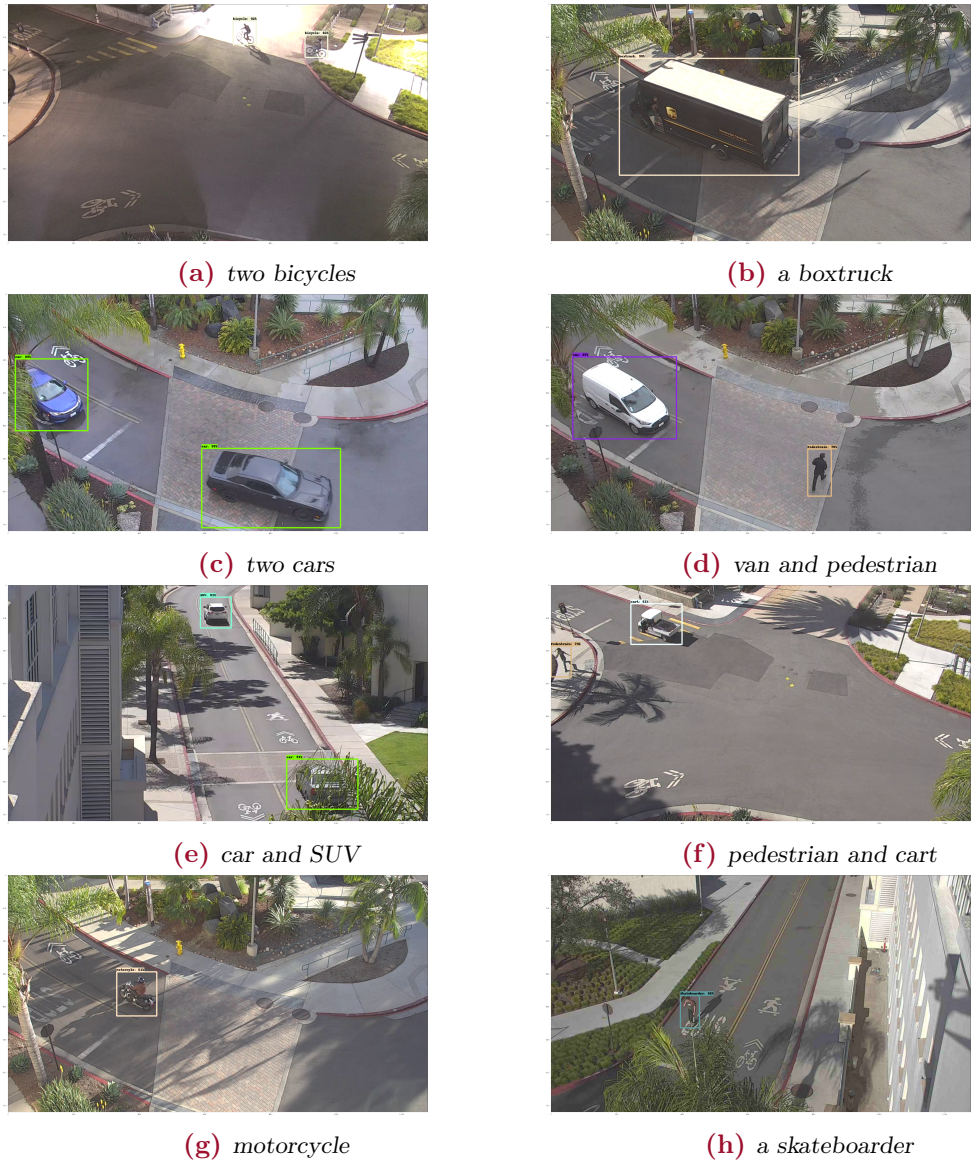
```
export DIR="/mnt/beegfs/home/yazdani/EE798/Project"
python3 exporter_main_v2.py --input_type image_tensor \
  --pipeline_config_path \
  "$DIR/models/Mobilenet_v2_all_300x300_0628" \
  --trained_checkpoint_dir \
  "$DIR/models/Mobilenet_v2_all_300x300_0628" \
  --output_directory "$DIR/exported-model/Mobilenet_v2_all_300x300_0628"
```

Note that in the command above, the path to the model directory is the directory that we make before training and contains the *pipeline.config* file. To create a TFLite model for deployment on an EdgeTPU device, we export the model using script *'export_tflite_graph_tf2.py'*

```
export DIR="/mnt/beegfs/home/yazdani/EE798/Project"
python3 export_tflite_graph_tf2.py --pipeline_config_path \
  "$DIR/models/Mobilenet_v2_all_300x300_0628" \
  --trained_checkpoint_dir \
  "$DIR/models/Mobilenet_v2_all_300x300_0628" \
  --output_directory "$DIR/exported-model-tflite/Mobilenet_v2_all_300x300_0628"
```

The exported model from this command is an interim model and we cannot run inferencing using this model. In the next section, we use this interim model for quantization and

conversion to a TFLite model. In Figure 5.1, detection on images in the test data set for different labels using the exported model are plotted. This exported model is not quantized and can only run on a CPU. The confidence threshold is set to 0.4 for these detections.



(a) *two bicycles* **(b)** *a boxtruck*

(c) *two cars* **(d)** *van and pedestrian*

(e) *car and SUV* **(f)** *pedestrian and cart*

(g) *motorcycle* **(h)** *a skateboarder*

Figure 5.1: *Detection of different objects at different camera pan and tilt angles, including multiple object detections in a single frame. These detections were performed using the interim exported model on a CPU.*

5.2 Quantization and TFlite Conversion

Quantization is performed to convert CNN parameters and weights with float32 data types to either int8 (8-bit signed integer in the decimal range [-128,..,127]) or uint8 (8-bit unsigned integer in the decimal range [0,..,255]). In this project we picked the uint8 data type. In order to perform quantization on a model, a representative dataset is needed. A representative dataset is a group of images on which a TFLite model is expected to inference. This is a fairly small dataset to calibrate or estimate the minimum and maximum of all floating-point arrays in a model. A few hundred sample images need be randomly chosen for this step, however, we used all captured images as the representative dataset. A v1 compatible version of TensorFlow was used for quantization; we found multiple issues with quantization under TF2 (TensorFLow v2).

```
import cv2
import random
import glob
import io
import os
import numpy as np
import tensorflow.compat.v1 as tf
```

Figure 5.2: *The TensorFlow compat.v1 module was used for quantization.*

We use OpenCV to read an image and the convert it to a tensor; this is done by converting the image into a uint8 numpy array followed by normalization (division by 255):

```
counter=0
max_num_rep_date=16758
total_images_tensor=np.empty([max_num_rep_date,300,300,3])
for filename in glob.glob('../Project/images/*.PNG'):
        img=cv2.imread(filename)
        img=cv2.cvtColor(cv2.resize(img,(300,300)),cv2.COLOR_BGR2RGB)
        if counter< max_num_rep_date:
            image_np=np.uint8(img)/255   # /255 should be in
            exp_img_np=np.expand_dims(image_np, axis=0)
            total_images_tensor[counter] = exp_img_np
            counter+=1
        else:
              break


def representative_dataset():
    for data in tf.data.Dataset.from_tensor_slices((total_images_tensor)).batch(1).take(16758):
        yield [tf.dtypes.cast(data, tf.float32)]
```

The following code is used to convert the interim exported model to TFLite:
After the TFLite model is generated, the final step is compiling it for an EdgeTPU device using the EdgeTPU compiler:

```
edgetpu_compiler Mobilenet_v2_all_300x300_0628_quant_all_images.tflite
```

The edgetpu compiler invocation will generate a tflite model named

```
converter = tf.lite.TFLiteConverter.from_saved_model('../Mobilenet_v2_all_300x300_0628/saved_model/')
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.allow_custom_ops = True
converter.representative_dataset = representative_dataset
converter.target_spec.supported_types = [tf.int8]
converter.inference_input_type = tf.uint8
converter.inference_type = tf.uint8
converter.post_training_quantize = True
tflite_quant_model = converter.convert()
TFLITE_FILE_PATH = '../Mobilenet_v2_all_300x300_0628/Mobilenet_v2_all_300x300_0628_quant_all_images.tflite'
with open(TFLITE_FILE_PATH, 'wb') as f:
  f.write(tflite_quant_model)
```

`Mobilenet_v2_all_300x300_0628_quant_all_images_edgetpu.tflite`

The _*edgetpu* model is executable by the TPU on the board. The compiler also generates a compilation log which enumerates all operations mapped to the TPU. An example of this log is shown in Figure 5.3.

```
Edge TPU Compiler version 15.0.340273435
Input: Mobilenet_v2_all_300x300_0628_quant_all_images.tflite
Output: Mobilenet_v2_all_300x300_0628_quant_all_images_edgetpu.tflite

Operator                       Count      Status

DEQUANTIZE                     2          Operation is working on an unsupported data type
QUANTIZE                       12         Mapped to Edge TPU
CONV_2D                        55         Mapped to Edge TPU
DEPTHWISE_CONV_2D              17         Mapped to Edge TPU
LOGISTIC                       1          Mapped to Edge TPU
RESHAPE                        13         Mapped to Edge TPU
CONCATENATION                  2          Mapped to Edge TPU
CUSTOM                         1          Operation is working on an unsupported data type
ADD                            10         Mapped to Edge TPU
```

Figure 5.3: *Example edgetpu compiler log.*

In this log, number of each operation mapped to the EdgeTPU is reported. Using a tool like Netron, we are able to see the layout of this model in Figure 5.4.
Input of the TFLite model is a tensor of data type uint8. Outputs are shown at the bottom of this image and they are: *scores*, *object type*, *bounding boxes* and *maximum number of detections*; these parameters are of data type float32 (single-precision floating-point).

5.3 Object Detection Using TFLite Model

To read and use the converted TFLite model on an EdgeTPU board, we need to invoke the model within a Python script. The tflite run-time module is imported in Python using following code

```
import tflite_runtime.interpreter as tflite
from pycoral.utils.edgetpu import make_interpreter
```

The following statement instantiates the interpreter

```
interpreter=make_interpreter
 ('../models/mobilenet_v2_All_300x300_0628_quant_all_images_edgetpu.tflite')
```

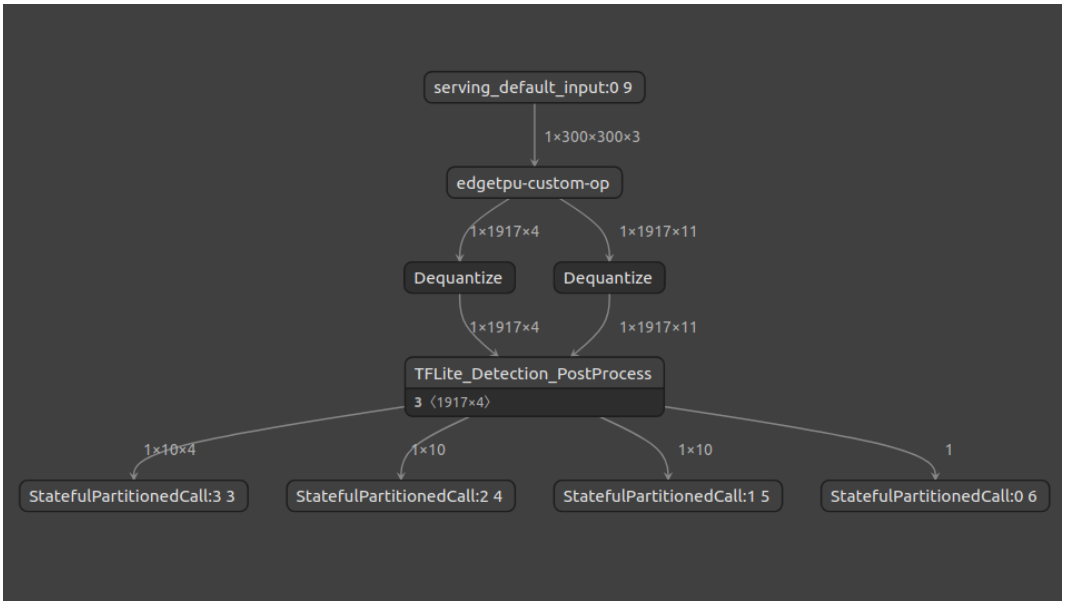
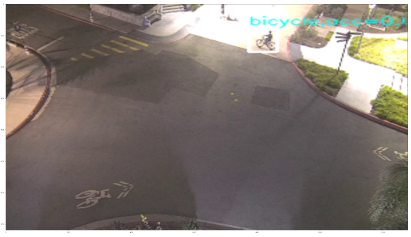


Figure 5.4: *TFLite model details in Netron.*

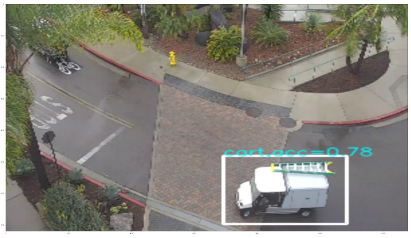
We then allocate a tensor for this interpreter, read images using OpenCV, convert each image into a tensor, and allocate each tensor to the interpreter to perform inferencing. In Figure 5.5, detections of different objects using TFLite model are shown. The confidence threshold is set to 0.4 for these detections.



(a) *bicycle*

(b) *bicycle*

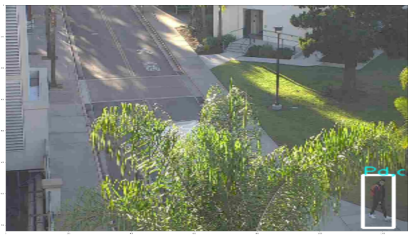
(c) *boxtruck*



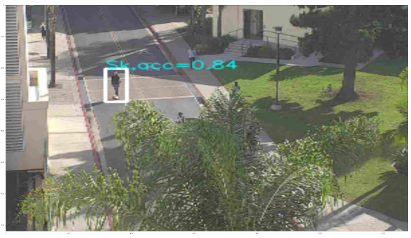
(d) *cart*

(e) *cart*

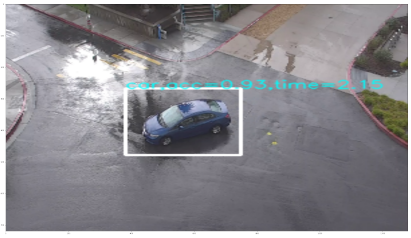
(f) *pedestrian*



(g) *pedestrian*



(h) *a skateboarder*



(i) *car*

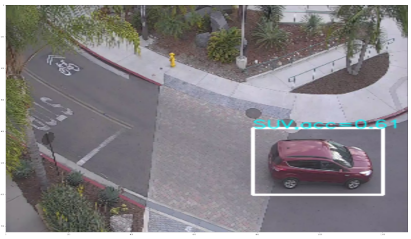
(j) *van*

(k) *truck*

(l) *two cars*

(m) *SUV (Sport Utility Vehicle)*

(n) *SUV (Sport Utility Vehicle)*



(o) *SUV (Sport Utility Vehicle)*

Figure 5.5: *Detection of different objects at multiple pan and tilt angles with the TFLite model executing on an EdgeTPU board.*

6 Results and Findings

In this section we discuss TFLite model performance on an EdgeTPU board. Relevant performance metrics were described in 2.2. We evaluate the performance of our developed TFLite model with IoU values of 0.5 and 0.75 and a confidence threshold of 0.4. The *ssd_mobilenet_v2_keras* model here refers to the TFLite model that has been generated in this work (*mobilenet_v2_All_300x300_0628_quant_all_images_edgetpu.tflite*). In addition, results from a TFLite model converted from a MobilenetV2 with FPN feature extractor (explained in sections 2.1 and 4.1), are also shown in Table 6.1 for comparison. As mentioned before, MobilenetV2 FPN is not suitable for real-time implementation due to computation complexity and speed. Results were obtained from running these models on an EdgeTPU board.

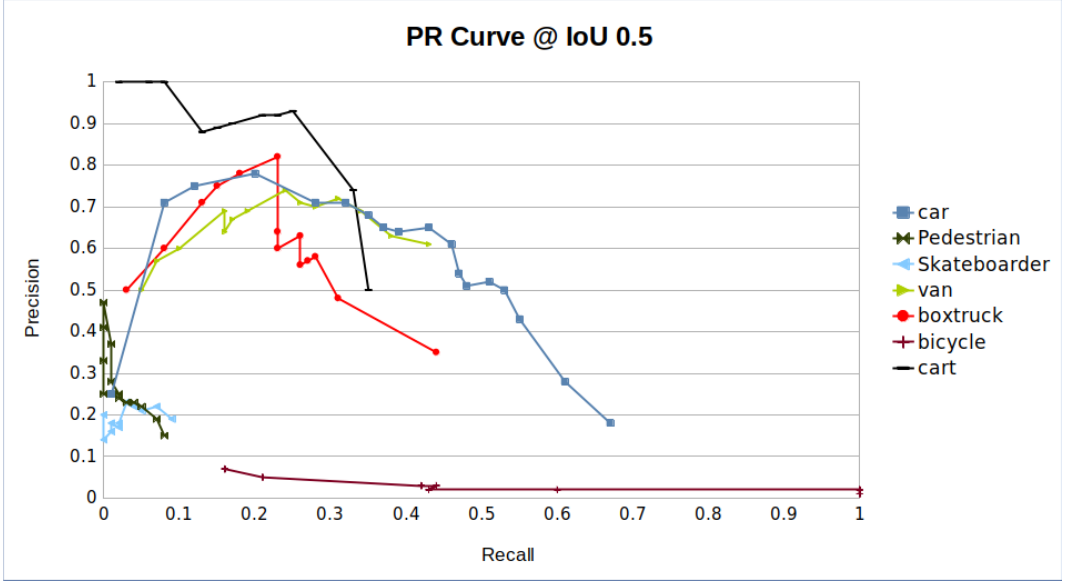


Figure 6.1: *Precision-Recall (PR) Curve at IoU 0.5 for different classes using the* ssd_mobilenet_v2_keras *model.*

For comparison, PR curves @IoU 0.75 for the FPN network is shown in Figure 6.3. We see that the AP of this network for different classes is better than the *ssd_mobilenet_v2_keras* network; though the FPN network cannot be used in real-time inferencing.
AP (Average Precision), which is the area under the PR curve, is higher for different classes at IoU 0.5 compared to IoU 0.75 (Figures 6.1 and 6.2). A summary of the performance of the TFLite model is presented in Table 6.1. As mentioned before, the *ssd_mobilenet_v2_keras* model is a good candidate for real-time object detection.

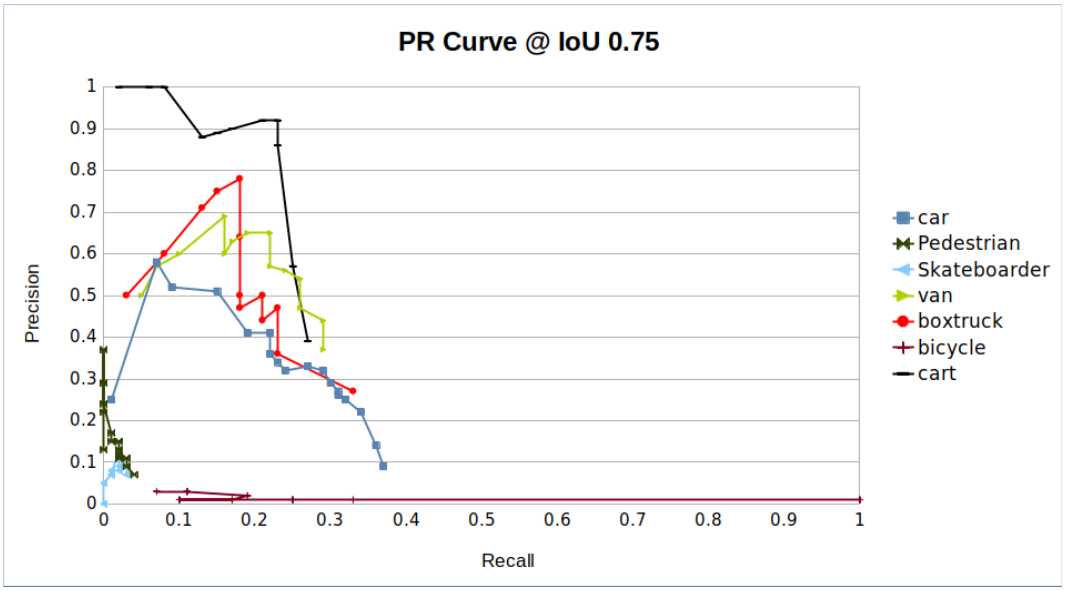


Figure 6.2: *Precision-Recall (PR) Curve at IoU 0.5 for different classes using* ssd_mobilenet_v2_keras *model*

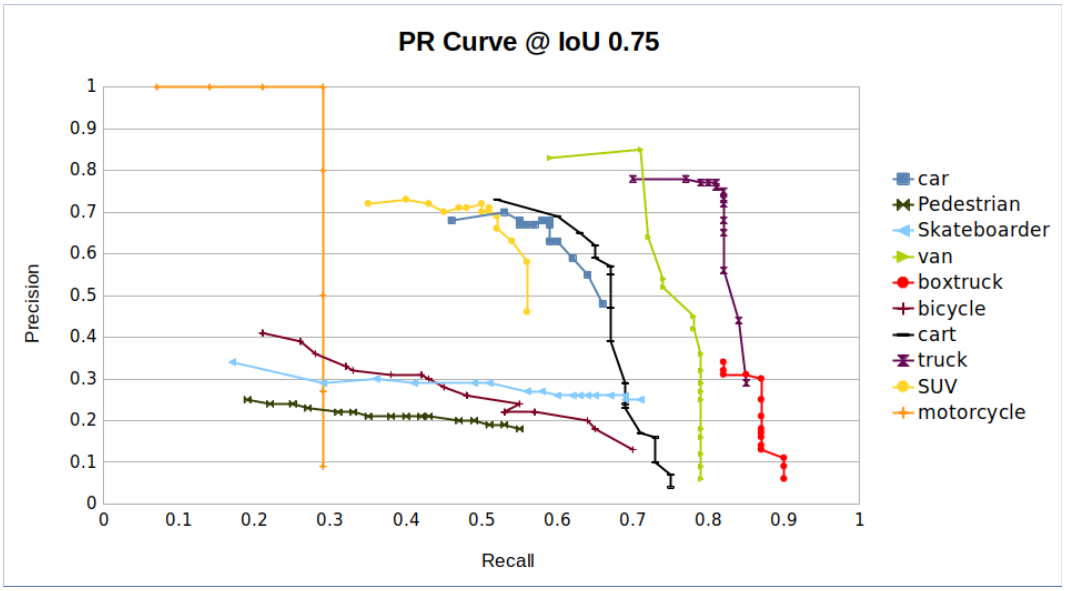


Figure 6.3: *Precision-Recall (PR) Curve at IoU 0.75 for different classes using* ssd_mobilenet_v2_fpn_keras *model*

This model runs inferencing in typically 8 ms. The mAP values related to this model performance are reported for IoU 0.5 and IoU 0.75. As expected, mAP drops as the IoU threshold increases. Comparing *ssd_mobilenet_v2_fpn_keras* with this model, we can easily notice better mAP which comes at a cost of computational complexity. The

Table 6.1: *Performance Metrics of TFLite Models*

Model Type	ssd_mobilenet_v2_keras	ssd_mobilenet_v2_fpn_keras
Frame Size	300x300	640x640
Inference Speed	8 ms	390 ms
mAP @IoU0.5	0.17	–
mAP @ IoU0.75	0.1	0.51

operations in the fpn model cannot be fully mapped to the TPU device, and as a result some of the operations are performed on CPU which makes this model very slow (typically 390 ms inferencing time) compared to the *ssd_mobilenet_v2_keras* model.

7 Conclusion

In this project, real-time object detection on different vehicles, pedestrians, and skateboarders has been done using a MobilenetV2 Convolutional Neural Network model. This model was trained on custom images that were captured from a live stream camera located on the 6th story balcony of the GMCS building at SDSU that monitors a three-way intersection on campus. The model is then converted into a lightweight version (called TFLite) that can be implemented on a Coral EdgeTPU board. The procedure to capture the images, annotate them, training the model, performing quantization, conversion to a TFLite model, and compiling it to be used on EdgeTPU are explained in this report. The results in terms of object detection performance metrics are also reported.

After observing performance of the TFLite model on test images and a live stream we observed that the performance of the model is better on objects with no shadows; Therefore, objects on an overcast day or illuminated nighttime have higher chance of being detected. In addition, the TFLite model detects larger objects better than smaller objects like pedestrians. There is a trade-off between detection of smaller objects and speed as discussed in Section 2.1.

In order to improve detection performance, future efforts would include

- Training a MobileNetV3 network with same image dataset in this work; the V3 model is expected to have improved accuracy.
- Obtaining more images from smaller objects like bicycles and motorcycles and possibly introducing new objects like scooters.

In the near future, we plan to compute Safety Surrogate Measure (SSM) parameters of interest like Time-To-Collision (TTC) and Post Encroachment Time (PET) in real-time using the TFLite model generated in this work.

References

[1] Erfan Chowdhury Shourov Christopher Paolini. "Detection and Classification of Pedestrians and Skateboarders for Computation of Surrogate Safety Measures (SSM)". In: *IEEE IV2020* (2020).

[2] Seth Cutter. *Analyzing the Potential of Geofencing for Electric Bicycles and Scooters in the Public Right of Way*. 2020.

[3] Nikolaos Doulamis Athanasios Voulodimos. "Deep Learning for Computer Vision: A Brief Review". In: *Hindawi* (2018).

[4] Shou-tao Xu Zhong-Qiu Zhao. "Object Detection with Deep Learning: A Review". In: *IEEE* (2019).

[5] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Dec. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[6] D. Erhan W. Liu D. Anguelov. "SSD: Single shot multibox detector". In: *European conference on computer vision* (2016).

[7] Bo Chen Andrew G. Howard Menglong Zhu. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *Google Inc.* (2017).

[8] URL: https://coral.ai/products/dev-board/.

[9] Jonathan Hui. *Understanding Feature Pyramid Networks for object detection (FPN)*. Mar. 2018. URL: https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c.

[10] P. Kim Z. Fang. "Comparison of Deep-Learning Algorithms for the Detection of Railroad Pedestrians". In: *Journal of information and communication convergence engineering* (2020).

[11] Soumik Rakshit. *Intersection Over Union*. Apr. 2019. URL: https://medium.com/koderunners/intersection-over-union-516a3950269c.

[12] Abdou Rockikz. *How to Perform YOLO Object Detection using OpenCV and PyTorch in Python*. Jan. 2020. URL: https://www.thepythoncode.com/article/yolo-object-detection-with-opencv-and-pytorch-in-python.

[13] Ren Jie Tan. *Breaking Down Mean Average Precision (mAP)*. Mar. 2019. URL: https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52.

[14] Suramya Tomar. "Converting video formats with FFmpeg". In: *Linux Journal* 2006.146 (2006), p. 10.

[15] A. Dutta, A. Gupta, and A. Zissermann. *VGG Image Annotator (VIA)*. http://www.robots.ox.ac.u 2016.

[16] Abhishek Dutta and Andrew Zisserman. "The VIA Annotation Software for Images, Audio and Video". In: *Proceedings of the 27th ACM International Conference on Multimedia*. MM '19. Nice, France: ACM, 2019. DOI: 10.1145/3343031.3350535. URL: https://doi.org/10.1145/3343031.3350535.

[17] Chowdhury E Shourov and Christopher Paolini. *Skateboarder and Pedestrian Conflict Zone Detection Dataset*. Nov. 2020. DOI: 10.17605/OSF.IO/NYHF7. URL: osf.io/nyhf7.

[18] Christopher Paolini et al. *Skateboarder and Pedestrian Dataset*. Mar. 2020. DOI: 10.17605/OSF.IO/CQD9Z. URL: osf.io/cqd9z.

[19] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[20] *https://www.tensorflow.org/tutorials/images/transfer_learning*.

[21] URL: https://medium.com/swlh/creating-your-own-custom-object-detector-using-transfer-learning-f26918697889.

[22] URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md.

[23] Mauro Castelli Ibrahem Kandel. "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset". In: *ICT Express* (2020).

A Appendix A: model_main_tf2.py

```
# Lint as: python3
# Copyright 2020 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================

r"""Creates and runs TF2 object detection models.

For local training/evaluation run:
PIPELINE_CONFIG_PATH=path/to/pipeline.config
MODEL_DIR=/tmp/model_outputs
NUM_TRAIN_STEPS=10000
SAMPLE_1_OF_N_EVAL_EXAMPLES=1
python model_main_tf2.py -- \
  --model_dir=$MODEL_DIR --num_train_steps=$NUM_TRAIN_STEPS \
  --sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES \
  --pipeline_config_path=$PIPELINE_CONFIG_PATH \
  --alsologtostderr
"""
from absl import flags
import tensorflow.compat.v2 as tf
from object_detection import model_lib_v2

flags.DEFINE_string('pipeline_config_path', None, 'Path to pipeline config '
                    'file.')
flags.DEFINE_integer('num_train_steps', None, 'Number of train steps.')
flags.DEFINE_bool('eval_on_train_data', False, 'Enable evaluating on train '
                  'data (only supported in distributed training).')
flags.DEFINE_integer('sample_1_of_n_eval_examples', None, 'Will sample one of '
                     'every n eval input examples, where n is provided.')
flags.DEFINE_integer('sample_1_of_n_eval_on_train_examples', 5, 'Will sample '
                     'one of every n train input examples for evaluation, '
                     'where n is provided. This is only used if '
                     '`eval_training_data` is True.')
```

```
flags.DEFINE_string(
    'model_dir', None, 'Path to output model directory '
                       'where event and checkpoint files will be written.')
flags.DEFINE_string(
    'checkpoint_dir', None, 'Path to directory holding a checkpoint.  If '
    '`checkpoint_dir` is provided, this binary operates in eval-only mode, '
    'writing resulting metrics to `model_dir`.')

flags.DEFINE_integer('eval_timeout', 3600, 'Number of seconds to wait for an'
                     'evaluation checkpoint before exiting.')

flags.DEFINE_bool('use_tpu', False, 'Whether the job is executing on a TPU.')
flags.DEFINE_string(
    'tpu_name',
    default=None,
    help='Name of the Cloud TPU for Cluster Resolvers.')
flags.DEFINE_integer(
    'num_workers', 1, 'When num_workers > 1, training uses '
    'MultiWorkerMirroredStrategy. When num_workers = 1 it uses '
    'MirroredStrategy.')
flags.DEFINE_integer(
    'checkpoint_every_n', 1000, 'Integer defining how often we checkpoint.')
flags.DEFINE_boolean('record_summaries', True,
                     ('Whether or not to record summaries during'
                      ' training.'))

FLAGS = flags.FLAGS


def main(unused_argv):
  flags.mark_flag_as_required('model_dir')
  flags.mark_flag_as_required('pipeline_config_path')
  tf.config.set_soft_device_placement(True)

  if FLAGS.checkpoint_dir:
    model_lib_v2.eval_continuously(
        pipeline_config_path=FLAGS.pipeline_config_path,
        model_dir=FLAGS.model_dir,
        train_steps=FLAGS.num_train_steps,
        sample_1_of_n_eval_examples=FLAGS.sample_1_of_n_eval_examples,
        sample_1_of_n_eval_on_train_examples=(
            FLAGS.sample_1_of_n_eval_on_train_examples),
        checkpoint_dir=FLAGS.checkpoint_dir,
        wait_interval=300, timeout=FLAGS.eval_timeout)
```

```
  else:
    if FLAGS.use_tpu:
      # TPU is automatically inferred if tpu_name is None and
      # we are running under cloud ai-platform.
      resolver = tf.distribute.cluster_resolver.TPUClusterResolver(
          FLAGS.tpu_name)
      tf.config.experimental_connect_to_cluster(resolver)
      tf.tpu.experimental.initialize_tpu_system(resolver)
      strategy = tf.distribute.experimental.TPUStrategy(resolver)
    elif FLAGS.num_workers > 1:
      strategy = tf.distribute.experimental.MultiWorkerMirroredStrategy()
    else:
      strategy = tf.compat.v2.distribute.MirroredStrategy()

    with strategy.scope():
      model_lib_v2.train_loop(
          pipeline_config_path=FLAGS.pipeline_config_path,
          model_dir=FLAGS.model_dir,
          train_steps=FLAGS.num_train_steps,
          use_tpu=FLAGS.use_tpu,
          checkpoint_every_n=FLAGS.checkpoint_every_n,
          record_summaries=FLAGS.record_summaries)

if __name__ == '__main__':
  tf.compat.v1.app.run()
```

B Appendix B: partition_dataset.py

```
#Lint as :python3
""" usage: partition_dataset.py [-h] [-i IMAGEDIR] [-o OUTPUTDIR]
[-r RATIO] [-x]

Partition dataset of images into training and testing sets

optional arguments:
  -h, --help            show this help message and exit
  -i IMAGEDIR, --imageDir IMAGEDIR
                        Path to the folder where the image dataset is
                        stored. If not specified, the CWD will be used.
  -o OUTPUTDIR, --outputDir OUTPUTDIR
                        Path to the output folder where the train and test
                        dirs should be created. Defaults to the same
                        directory as IMAGEDIR.
  -r RATIO, --ratio RATIO
                        The ratio of the number of test images over the
                        total number of images. The default is 0.1.
  -x, --xml             Set this flag if you want the xml annotation
  files to be processed and copied over.
"""
import os
import re
from shutil import copyfile
import argparse
import math
import random
import pandas as pd
prjdir="/mnt/beegfs/home/yazdani/EE798/Project/"
annotdir="annotations/training_0530_all/"
csvfile="Final_annot_0530_Plus_Pedestrian_Skateboarder.csv"
final_annot=pd.read_csv(prjdir+annotdir+csvfile)
train_annotation=final_annot



os.system('rm -r /mnt/beegfs/home/yazdani/EE798/Project/images/train')
os.system('rm -r /mnt/beegfs/home/yazdani/EE798/Project/images/test')
def iterate_dir(source, dest, ratio, copy_xml):
    source = source.replace('\\', '/')
    dest = dest.replace('\\', '/')
    train_dir = os.path.join(dest, 'train')
    test_dir = os.path.join(dest, 'test')
```

```
    if not os.path.exists(train_dir):
        os.makedirs(train_dir)
    if not os.path.exists(test_dir):
        os.makedirs(test_dir)

    images = [f for f in os.listdir(source)
              if re.search(r'([a-zA-Z0-9\s_\\.\-\(\):])+(.jpg|.jpeg|.PNG)$', f)]

    num_images = len(images)
    num_test_images = math.ceil(ratio*num_images)

    for i in range(num_test_images):
        idx = random.randint(0, len(images)-1)
        filename = images[idx]
        #  here this code is added to delete rows with test image names from training annotatio
        for row in train_annotation['filename'].transpose().iteritems():
            if row[1]==filename:
                train_annotation.drop(row[0],axis=0,inplace=True)
                train_annotation.to_csv('/mnt/beegfs/home/
                yazdani/EE798/Project/annotations/Partitioned_Annotations/train_annotation.csv'
        copyfile(os.path.join(source, filename),
                 os.path.join(test_dir, filename))

#         if copy_xml:
#             xml_filename = os.path.splitext(filename)[0]+'.xml'
#             copyfile(os.path.join(source, xml_filename),
#                      os.path.join(test_dir,xml_filename))
        images.remove(images[idx])
    test_annotation=pd.read_csv('/mnt/beegfs/home/yazdani/EE798/Project/annotations/training_05
    for filename in images:

        for row in test_annotation['filename'].transpose().iteritems():
            if row[1]==filename:
                test_annotation.drop(row[0],axis=0,inplace=True)
                test_annotation.to_csv('/mnt/beegfs/home/yazdani/EE798/Project/annotations/Part
        copyfile(os.path.join(source, filename),
                 os.path.join(train_dir, filename))


#         if copy_xml:
#             xml_filename = os.path.splitext(filename_train)[0]+'.xml'
#             copyfile(os.path.join(source, xml_filename),
#                      os.path.join(train_dir, xml_filename))
```

```
def main():

    # Initiate argument parser
    parser = argparse.ArgumentParser(description="Partition dataset of images into tr
                                     formatter_class=argparse.RawTextHelpFormatter)
    parser.add_argument(
        '-i', '--imageDir',
        help='Path to the folder where the image dataset is stored. If not specified,
        type=str,
        default=os.getcwd()
    )
    parser.add_argument(
        '-o', '--outputDir',
        help='Path to the output folder where the train and test dirs should be creat
             'Defaults to the same directory as IMAGEDIR.',
        type=str,
        default=None
    )
    parser.add_argument(
        '-r', '--ratio',
        help='The ratio of the number of test images over the total number of images.
        default=0.1,
        type=float)
    parser.add_argument(
        '-x', '--xml',
        help='Set this flag if you want the xml annotation files to be processed and
        action='store_true'
    )
    args = parser.parse_args()

    if args.outputDir is None:
        args.outputDir = args.imageDir

    # Now we are ready to start the iteration
    iterate_dir(args.imageDir, args.outputDir, args.ratio, args.xml)


if __name__ == '__main__':
    main()
```

C Appendix C: TFLite_inference_test.py

```
#
# python3 TFLite_inference_test.py >/dev/null 2>&1
#
import numpy as np
import tflite_runtime.interpreter as tflite
import cv2
import matplotlib
import matplotlib.pyplot as plt
import time
import datetime #import date
import paho.mqtt.client as mqtt
import sys;
import logging;
import json;

def on_connect(client, userdata, flags, rc):
   if rc==0:
          client.connected_flag=True #set flag
          logging.debug("paho mqtt client connected ok")
   elif rc==5:
          logging.debug("paho mqtt client not connected, authentication failure")
          client.bad_connection_flag=True
   else:
          logging.debug("paho mqtt client not connected, returned code=%d",rc)
          client.bad_connection_flag=True

logging.basicConfig(filename='/tmp/tflite.log', level=logging.DEBUG)

client_name='EdgeTPU'
client = mqtt.Client(client_name)
host='130.191.161.21' # broker address
client.connected_flag=False
client.bad_connection_flag=False
client.on_connect=on_connect  # bind callback function
client.username_pw_set(username="starlab",password="starlab!")
client.connect(host, port=1883, keepalive=60, bind_address="")

client.loop_start()  #Start loop

while not client.connected_flag and client.bad_connection_flag: #wait in loop
    logging.debug("In wait loop")
    time.sleep(1)

logging.debug('client.bad_connection_flag: %r',client.bad_connection_flag)
```

```
logging.debug('client.connected_flag: %r',client.connected_flag)

#if not client.bad_connection_flag:
#    client.loop_stop()

msg = f"started"
topic = f"pelco/edgetpu"
result = client.publish(topic, msg)
status = result[0]
if status == 0:
   logging.debug(f"Send `{msg}` to topic `{topic}`")
else:
   logging.debug(f"Failed to send message to topic {topic}")
   sys.exit()

from pycoral.utils.edgetpu import make_interpreter
category_index = {
    1: {'id': 1, 'name': 'car'},
    2: {'id': 2, 'name': 'truck'},
    3: {'id': 3, 'name': 'SUV'},
    4: {'id': 4, 'name': 'cart'},
    5: {'id': 5, 'name': 'bicycle'},
    6: {'id': 6, 'name': 'motorcycle'},
    7: {'id': 7, 'name': 'boxtruck'},
    8: {'id': 8, 'name': 'van'},
    9: {'id': 9, 'name': 'Pd'},
    10: {'id': 10, 'name': 'Sk'},
}

def draw_rect(image,box):
       y_min= int(max(1, (box[0]* image.shape[0])))
       x_min= int(max(1, (box[1] *image.shape[1])))
       y_max = int(min(image.shape[0], (box[2]*image.shape[0])))
       x_max= int(min(image.shape[1],(box[3]*image.shape[1])))
       cv2.rectangle(image, (x_min,y_min), (x_max,y_max), (255,255,255), 2)
       #print (x_min,y_min,x_max,y_max)
       return x_min,y_min,x_max,y_max
# Load TFLite model and allocate tensors.

#interpreter = tflite.Interpreter(model_path="/home/mendel/EE798/models/model_edgetpu.tflite")#
interpreter=make_interpreter('/home/mendel/EE798/models/Mobilenet_v2_all_300x300_0628_quant_all
#interpreter=make_interpreter('/home/mendel/EE798/models/mobilenet_v1_1.0_224_l2norm_quant.tfli
#interpreter=make_interpreter('/home/mendel/google-coral/tflite/python/examples/detection/models
interpreter.allocate_tensors()
```

```
# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

#img = cv2.imread('/home/mendel/EE798/images/image425879.PNG')
#videocap=cv2.VideoCapture('/home/mendel/EE798/NOTOS/Camera\ Recordings/MP4_videos_or
videocap=cv2.VideoCapture('rtsp://yazdani:arya1234@sunray.sdsu.edu/stream2')
#videocap=cv2.VideoCapture('/home/mendel/EE798/videos/video1.mp4')

fourcc=cv2.VideoWriter_fourcc('m','p','4','v')
out = cv2.VideoWriter('filename.avi',fourcc, 5, (1280,720),isColor=True)

def getFrame(sec):
    hasFrames,image=videocap.read()
    return hasFrames,image

sec = 0
frameRate=0.033
success= getFrame(sec)[0]

while success:
    sec=sec+frameRate
    sec=round(sec,2)
    success=getFrame(sec)[0]
    pre_time_start=time.perf_counter()
    #out.write(getFrame(sec)[1])
    img=getFrame(sec)[1]
    original_image=img
    img=cv2.cvtColor(cv2.resize(img , (300,300)), cv2.COLOR_BGR2RGB)
    ximg=np.uint8(img)
    #img=img/255.0
    #ximg=np.float32(img)#/255.0
    #ximg=np.uint8(img)
    exp_img=np.expand_dims(ximg, axis=0)
    interpreter.set_tensor(input_details[0]['index'], exp_img)
    pre_time_end=time.perf_counter()
    #for i in range (0,20):
    start=time.perf_counter()
    interpreter.invoke()
    inference_time=time.perf_counter()-start
    preprocess_time=pre_time_end-pre_time_start
```

```
output_data = interpreter.get_tensor(output_details[0]['index'])
font= cv2.FONT_HERSHEY_SIMPLEX
fontScale=0.3
color=(0,0,255)
thickness=1
scores=np.array(interpreter.get_tensor(output_details[2]['index']))
#print('inference time is:',format(inference_time))
#print('preprocess time is:',format(preprocess_time))
for index,score in enumerate(scores[0]):
    if score>0.4:
        box=np.array(interpreter.get_tensor(output_details[0]['index']))[0][index]
        classes=np.array(interpreter.get_tensor(output_details[1]['index']))[0][index]
        draw_rect(img,box)
        class_text=str("{},acc={:.2f},time={:.2f} sec" .format((category_index[classes+1]['
        org=(draw_rect(img,box)[0],draw_rect(img,box)[1])
        img=cv2.putText(img,class_text,org,font,fontScale,color,thickness,cv2.LINE_AA)
        x_min_disp=int(max(1, (box[1] *original_image.shape[1])))
        y_min_disp=int(max(1, (box[0]* original_image.shape[0])))
        x_max_disp=int(min(original_image.shape[1],(box[3]*original_image.shape[1])))
        y_max_disp=int(min(original_image.shape[0], (box[2]*original_image.shape[0])))
        t=datetime.datetime.now()
        #score_data=np.round(score,2)
        data = {
               "class": format(category_index[classes+1]['name']),
               "box": [format(x_min_disp), format(y_min_disp), format(x_max_disp), format(y
               "date": format(t.month) + '/' + format(t.day) + '/' + format(t.year),
               "time": format(t.hour) + ':' + format(t.minute) + ':' + format(t.second),
               "score": "{:.2f}".format(score)
        }
       # print("{:.2f}".format(score))
        msg = json.dumps(data)
        topic = f"pelco/edgetpu"
        result = client.publish(topic, msg)
        status = result[0]
        if status == 0:
           logging.debug(f"Send `{msg}` to topic `{topic}`")
        else:
           logging.debug(f"Failed to send message to topic {topic}")

        #today = date.today()
        #print('date:',format(today))
        #print('time:',format())
        #print(box)
#print(scores)
```

```
    # uncomment next four lines if not running as a service
    #
    img=cv2.cvtColor(cv2.resize(img, (1280,720)),cv2.COLOR_RGB2BGR)
    cv2.imshow('Live Pelco',img)
    out.write(img)
    cv2.waitKey(1)
    if (cv2.waitKey(1) & 0xFF == ord("q")):
        break

out.close()
cv2.destroyAllWindows()
client.loop_stop()
```