Object Detection using Raspberry Pi

CS230-Spring 2019 Course Project

Ranga Chadalavada | rangach@stanford.edu

Project Report

Introduction

The idea is to do object detection on Raspberry Pi. Using the Raspberry Pi's camera take a picture, send it through the object detection neural network running on Pi and detect the objects in the picture. Raspberry Pi costs less than $100.

Why Raspberry Pi

The Raspberry Pi 3 Model B is a tiny credit card size computer. It has a Quad core 64-bit processor clocked at 1.4GHz with 1GB SRAM. It also has Dual-band 2.4GHz and 5GHz wireless LAN and High speed Ethernet up to 300Mbps. It runs Raspbian OS, which is similar to Linux. It is just a motherboard. What makes Raspberry Pi interesting is that it has a number of sensors available for it, like Temperature, Humidity, Air Pressure, Gas, Motion Sensors, Navigation Modules, Motors, etc.

Objective of the project

There are a couple of objective:

1. To familiarize myself with raspberry pi and understand its capabilities.

2. To understand object detection and theory behind it.

3. To evaluate and select a model based on the systems capability.

4. Modify the model by enhancing to detect additional classes and partly retrain it.

Basis for the idea for the project

The idea of doing this project came after going over a number of articles as listed below:

1. How to easily Detect Objects with Deep Learning on Raspberry Pi

   https://medium.com/nanonets/how-to-easily-detect-objects-with-deep-learning-on-raspberrypi-225f29635c74

2. Real-Time Object Detection on Raspberry Pi Using OpenCV DNN

https://heartbeat.fritz.ai/real-time-object-detection-on-raspberry-pi-using-opencv-dnn-98827255fa60

3. Raspberry Pi: Deep learning object detection with OpenCV

https://www.pyimagesearch.com/2017/10/16/raspberry-pi-deep-learning-object-detection-with-opencv

4. Real-time Object Detection with MXNet On The Raspberry Pi

https://mxnet.incubator.apache.org/versions/master/tutorials/embedded/wine_detector.html

5. Raspberry Pi Face Recognition

https://www.pyimagesearch.com/2018/06/25/raspberry-pi-face-recognition/

Current Status

I tried installing MXNet and OpenCV but had trouble installing them. I was not able to find compatible versions installable on Raspberry Pi. However, I was able to install Tensorflow and Keras. So I decided to try an implementation based on Tensorflow.

I downloaded the code for an existing implementation from github (https://github.com/tensorflow/models).

Parts of this tutorial on utube (https://www.youtube.com/watch?v=wh7_etX91ls) was helpful is guiding through in installing the code.

I downloaded the models (frozen graphs) from Https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

I wrote a lightweight webserver which when a request is made takes a picture using the Pi's camera. The picture is fed to the object detection code (which is based on the implementation downloaded). The output is then sent back to the web requestor.

I tried out some models form the above mentioned repository and the results are shown below:

Model	Time to Process	Issues
ssd_mobilenet_v1_coco	6.91 sec	
ssd_mobilenet_v2_coco	7.93 sec	Memory overrun
ssd_mobilenet_v1_fpn_coco	67.56	Memory overrun
faster_rcnn_nas_coco	--	Memory insufficient

Based on an understanding that quantizing the model improved performance and reduced memory requirement, I tried to quantize one of the models but have been having issues with Tensorflow and Keras versions.

For now, I have decided for this project to go with ssd_mobilenet_v1_coco model.

Dataset

I have not used any dataset till now, but for the next step I may have to use one. I will select the data set based on availability of annotated images for object detection containing classes not previously processed by the model.

Code Link

https://github.com/rangach/object_detection_new

piodserver.py is file to be run. It will run only on Raspberry Pi, unless all references to PiCamera are commented out and the url for the server is changed to localhost.

Final Phase

I wanted to add additional class for detection to the existing trained model. I was looking at modifying the frozen graph (frozen_inference_graph.pb) but came across this article

https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/

which described steps for retraining the existing model, for detection of additional classes using the checkpoint files, that came along with the frozen graph (in this case ssd_mobilenet_v1_coco.tar.gz from Https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md)

The original model was trained on the coco dataset and was able to detect 90 classes as under:

Person, bicycle, car, Motorcycle, Airplane, Bus, Train, boat, traffic light, fire hydrant, stop sign, parking, meter, bench, Bird, Cat, Dog, Horse, Sheep, Cow, Elephant, bear, Zebra, giraffe, Backpack, Umbrella, Handbag, Tie, Suitcase, Frisbee, Skis, Snowboard, sports ball, Kite, baseball bat, baseball glove, Skateboard, Surfboard, tennis racket, Bottle, wine glass, cup, fork, knife, spoon, Bowl, Banana, Apple, Sandwich, Orange, Broccoli, Carrot, hot dog, Pizza, Donut, Cake, Chair, Couch, potted plant, Bed, dining table, Toilet, Tv, Laptop, Mouse, Remote, Keyboard, cell phone, Microwave, Oven, Toaster, Sink, Refrigerator, Book, Clock, Vase, Scissors, teddy bear, hair drier, Toothbrush

I added a new class ‘Pen’ and retrained the network.

Here are the steps taken to do so:

a) I downloaded 125 images of Pen’s (From articles on the web I had gathered that a set of 100 to 150 images would do a satisfactory job for retraining).

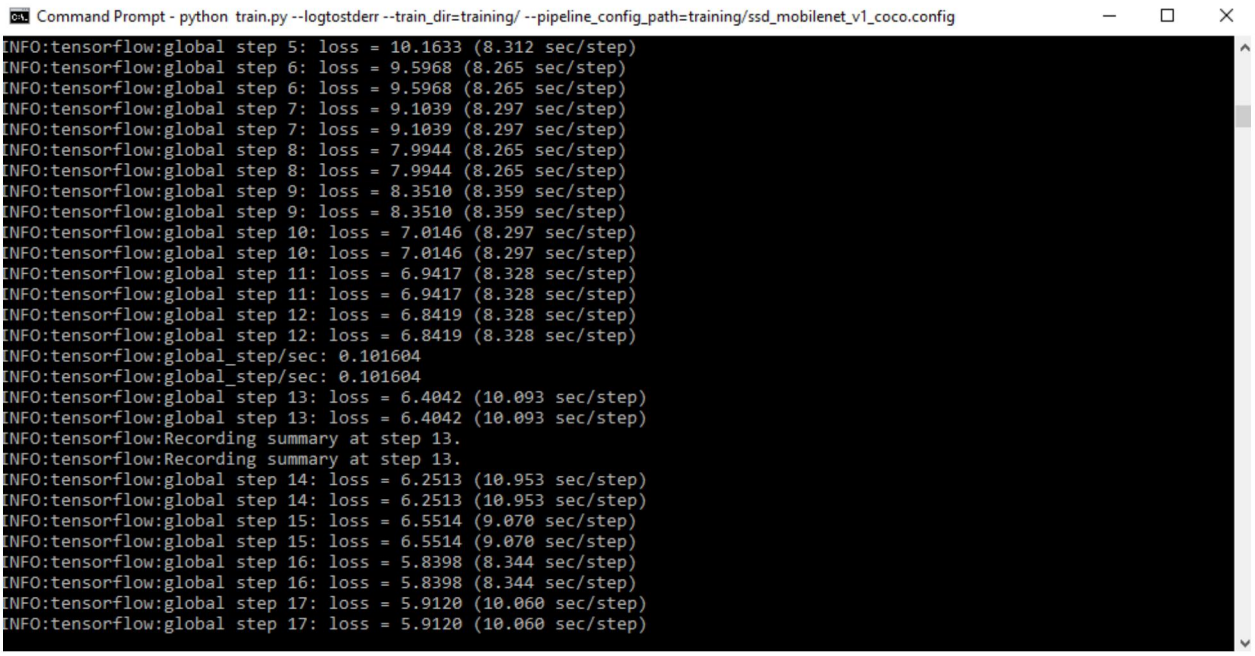
b) I divided them into a training set of 110 images and test set of 15 images. Having such a small set, I did not keep aside any for validation.
c) I used LabelImg to annotate and created the annotated xml files.
d) Converted the xml files to csv files.
e) Converted the csv files to TFRecords for training and test.

All the related code is in https://github.com/rangach/object_detection_new/Training-Tensorflow/

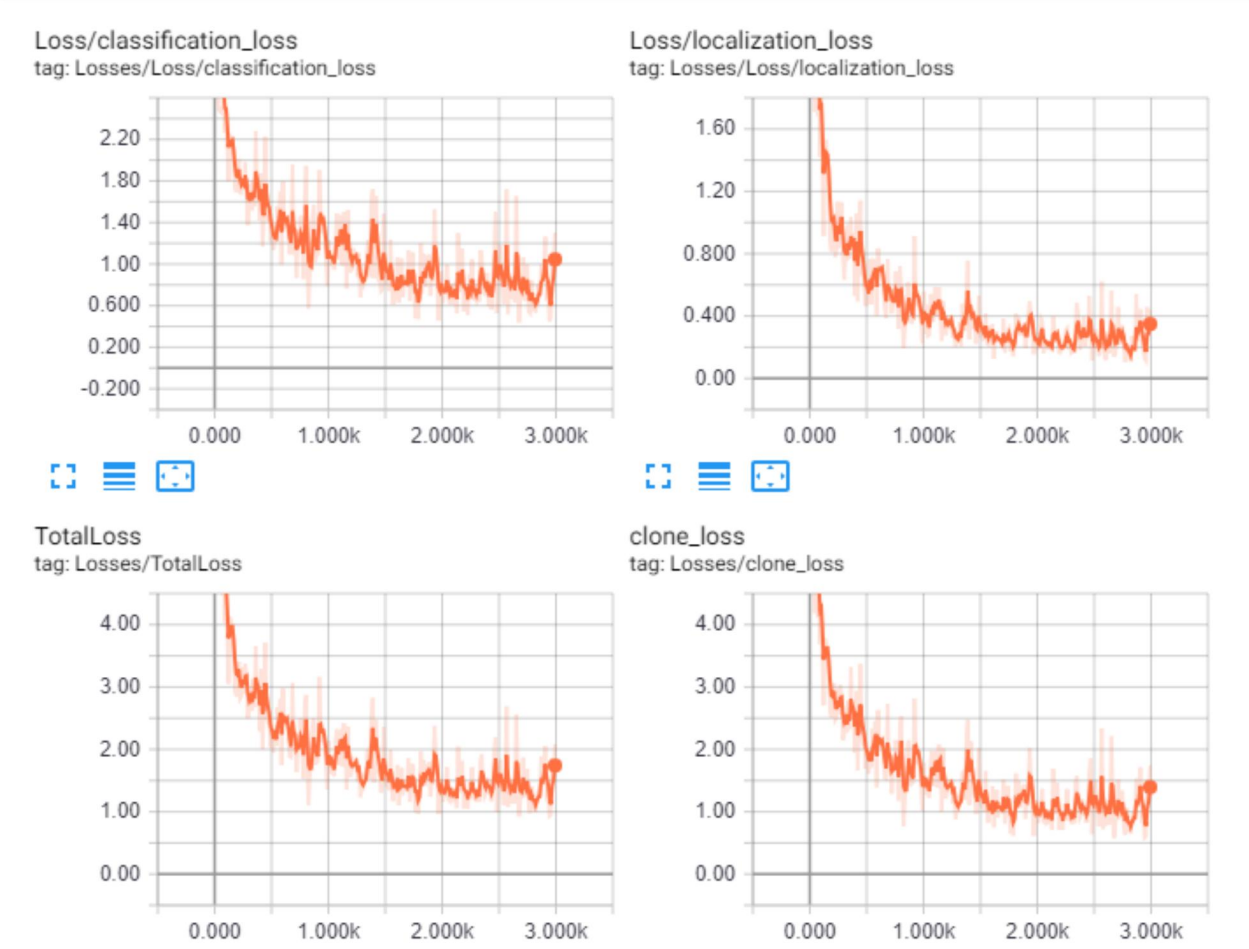
I modified the ssd_mobilenet_v1_coco.config file to point to the new TFRecords for training and test. I made it point to the new label_map.pbtxt file. I pointed to the model.ckpt file that came with the ssd_mobilenet_v1_coco.tar.gz. I changed the number of iterations for training to 3000. (In a previous test run I had achieved a loss of around two in about 2200 iterations. The article had mentioned a loss between one and two would be ideal. A loss below one for a small dataset may result in overfitting).

I used TensorFlow/models/research/object_detection/legacy/train.py (available on github) to train.

The training started with an initial loss of around 10 (shown below)



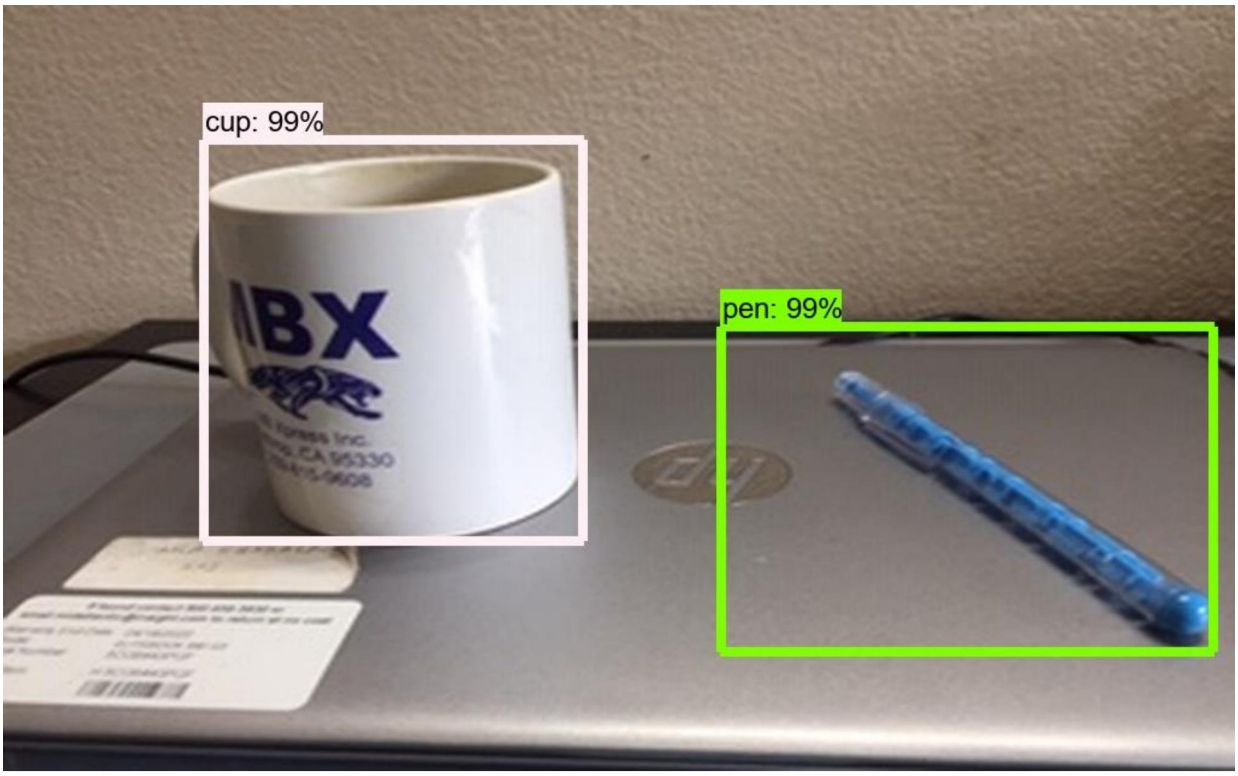
I trained for 3000 iterations on a windows PC (I did not think Raspberry PI was an apt platform for training). Training requires a lot more resources than runtime. At the end of training, the loss was between one and two. (Shown below)



Then using the final model.ckpt file that was generated during training and using

[TensorFlow/models/research/object_detection/export_inference_graph.py](), I generated the new frozen graph (frozen_inference_graph.pb) file, which now include parameters which are also trained for the new class (pen).

The model was able to detect the additional class (pen) that it was trained for (shown below)



Conclusion

I think the ability to run Neural Networks on such a low cost devices to accomplish tasks, will permeate their wide spread use for daily activities.