Object Detection on a Raspberry Pi Edge Platform with Azure Custom Vision AI

Ezana N. Beyenne

MSDS 462: Final Project

Professor Noah Gift

06/05/2021

Introduction and Problem Statement

This final project aims to build and deploy an edge-based computer vision solution using Azure Custom Vision AI to load, label, train, build a single-stage object detection model, and subsequently deploy it to a Raspberry Pi edge-based platform. The purpose of this project is to serve as a proof of concept to iteratively improve the model by deploying it in the real world and getting as real-time feedback as possible, starting with the detection of four labels, namely: 1) cars, 2) motorbikes, 3) person, and 4) potholes. The architecture of this Azure-developed model and Raspberry Pi 4 edge-based platform is shown below in Figure 1. To understand how far we must improve our edge-based solution, I compared it against the RetinaNet COCO 2017 object detection model.

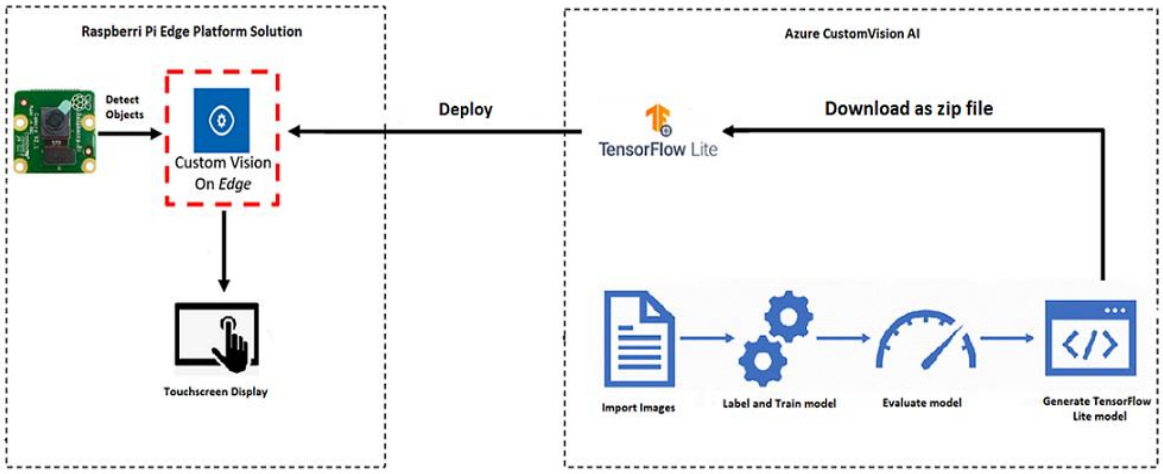

Figure 1. Raspberry Pi 4 Single-Stage Object Detection Edge-based platform

Data Collection

I collected data from the Kaggle site for my four labels: 1) cars, 2) motorbikes, 3) person and 4) potholes. I had an unbalanced dataset with 100 images for cars, 235 images for motorbikes, 249 images for people, and 113 images for potholes. I had to create the bounding boxes around each of these images manually. The "get suggested objects" feature did not do a good job labeling the untagged images or classifying them correctly. I had to spend additional time fixing these misidentifications and bounding boxes.

Training and Performance

Azure Custom Vision AI allows you to use "*Quick Training*" or "*Advanced Training*". The "*Advanced Training*" option allows you to specify a compute time budget, and it will experimentally choose the best training and augmentation techniques for your budget. The more time it spends training the model, the better the outcome. Since we are at the proof-of-concept stage of developing this touchscreen-assisted object detection platform, I used the "*Quick Training"* option. The site did not give me the option of downloading the machine learning code as a notebook, nor did it give me the option of splitting the dataset into the train, test, and validation partition. The precision, recall and mean average precision scores after the eleventh

iterative training is shown below in Figure 2.

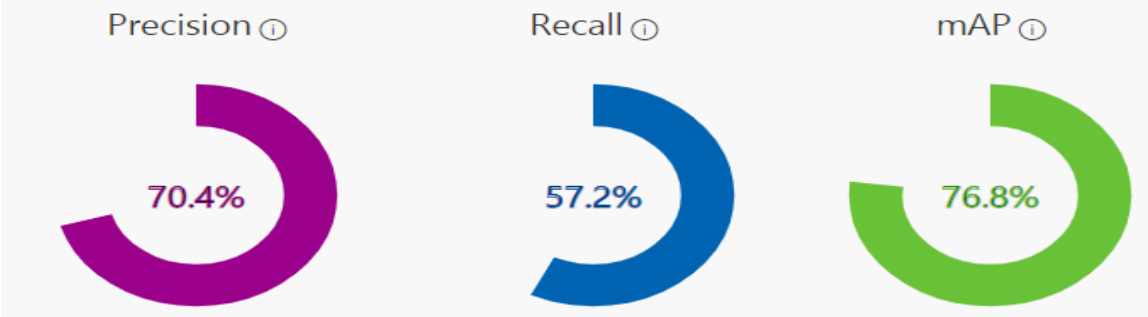

Figure 2. Performance after the 11th iteration

Edge-Based Platform

The edge-based platform consists of a Raspberry Pi 4 Module B, a Raspberry camera, and a Raspberry Pi 7-inch touchscreen. When initially building this platform, it overheated after a few minutes. I added heatsinks on the board processors and added a fan to the casing. These two solutions resolve the overheating issues. I then modified the predict.py code to handle the Raspberry Pi camera and annotate the bounding boxes on the touchscreen. In the RetinaNet model, I generated the TensorFlow Lite and COCO label files. I then modified the detect_objects.py file to load those files and provide a hardcoded threshold instead of generating the files passing them as an argument. These changes allowed me to load and run the platform as quickly and hassle-free as possible while running the heads-up display in the car.

Azure Custom Vision AI-generated model vs. RetinaNet COCO 2017 model

I downloaded the RetinaNet Object Detection TensorFlow Lite model and its eighty-nine labels to set a performance benchmark. I modified the camera code to load the RetinaNet COCO 2017 model and label files. Both models could identify cars and people, but the RetinaNet had tighter bounding boxes and better accuracy on the identifications. The Azure Custom Vision AI-generated model had problems identifying multiple objects from a distance with bounding boxes drawn slightly off. The differences between the two object detection systems were noticeable, as shown in the demo video, pointing to the Azure model needing more data, more accurate bounding box tagging during the tagging phase, and a longer training time.

Future Considerations

Additional data for each of the existing four labels would fix the precision and recall. More time fixing the bounding boxes of each of these training images and using the "*Advanced Training*" option for more training compute time would deliver better results. I could get better object detection results with a wide-angled camera that has zoom and night-vision capabilities. I could add more labels and road signs and lane detection capabilities, replace the touchscreen with a clear heads-up display attached to the front window and include speech recognition. As we add more labels, I could use transfer learning and model chaining instead of Custom Vision AI. Incorporating these changes would improve the model for better detection of the objects in real-time using edge-based computer vision solutions.

References

1. Kaggle Pothole Dataset: https://www.kaggle.com/sachinpatel21/pothole-image-dataset
2. Horses or Human Dataset: https://www.kaggle.com/sanikamal/horses-or-humans-dataset
3. Natural Images Dataset: https://www.kaggle.com/prasunroy/natural-images
4. Object Detection with RetinaNet: https://keras.io/examples/vision/retinanet/
5. Azure Custom Vision AI: https://www.customvision.ai/
6. GitHub repository for Final Project: https://github.com/Ezbze/NorthwesternU/tree/main/MSDS462/FinalProject/