Smart Parking Detection using Edge Technology on Google Coral Dev Board

Samayak Malhotra, Binoy George

Computer Science, Birla Institute of Technology and Science, Dubai, U.A.E

f20160054@dubai.bits-pilani.ac.in, binoymbinoy@gmail.com

***Abstract*— In this project Google Coral Dev board was used for providing accurate information to the user about the available parking spaces in a parking lot. Since it employs the latest Edge TPU technology, fast-processing machine learning was possible. For the purposes of testing this project, a small-sized prototype of the modern parking lot was created, consisting of parking spaces and toy cars. Six hundred pictures of the parking model were taken with and without cars from different angles. Image augmentation was done to increase the dataset size to 3919 images for training and testing purposes. Dataset training was done using Google Colab. Later, an inference script was written that was used for communication between the host machine and the Coral board. Web app interface was created using Javascript, HTML and CSS to display the working of our model. The final model accuracy achieved by our model was 100% indicating that our model can work successfully with real cars in parking lots outdoor.**

Keywords: Google coral, Parking Space Detection, Edge TPU, Coral camera, IoT, Machine learning, Transfer learning, Web app interface

I. Introduction

In today's world, road travel seems to be the most preferred choice for people whether it's a private or shared vehicle. A larger population correlates to a larger number of vehicles on the road leading to high carbon footprint. There has been an increase in the usage of private vehicles in urban regions, and it has been noticed that a lot of people face problems finding a parking, mainly in the malls during the peak hours. Currently, most of the existing car parks do not have a systematic parking system. Most of them are manually managed and is a little inefficient, providing inaccurate information to the user. We plan on designing a smart parking space detection system that could be used outdoors and in the malls allowing people to find the nearest parking available.

The problem that always occurs at the car park is the time that's being wasted in looking for available parking slot. Users keep circling the parking area until they find an empty parking space. This is due to the lack of implementation in technologies which are available in the market today. Car park entrances are controlled by barrier gates whereby parking tickets are used extensively for access purpose. With the growth of technology, these systems can be simplified in many ways.

P.Ajitha et al. in their paper [8] proposed method of reducing time in finding a parking. However their research lacks practical evidence that proves their method. On the other hand, our model is mainly focused on reducing the time taken in finding a parking spot and in turn reducing the carbon footprint as less fuel would be used while looking for a parking spot. With the help of the Edge TPU technology we would be able to process the data much faster as compared to other systems that are already available in the market.

The Coral Dev Board contains an Edge TPU coprocessor that's ideal for prototyping new projects that demand fast on-device inferencing for ML projects [1]. It would be able to provide us with fast and real-time outputs.The Edge TPU coprocessor is capable of performing 4 trillion operations per second (TOPS). For example, it can execute state-of-the-art mobile vision models such as MobileNet v2 at almost 400 frames per second, in a power efficient manner. This board comes with System-on-a-Chip, Machine Learning, Wifi connectivity and supports TensorFlow lite models and AutoML vision edge. With Edge Computing, data processing takes place on the device or the host machine instead of a data center. Edge computing helps reducing the bandwidth, increasing network performance by reducing latency and providing resilience and maximizing performance and operational efficiency [4].

In order to set-up and access the board, flashing of the Mendel Linux would be required. After having the access to the serial console, various image classification models are run on the board.

II. Literature Review

A. Coral Dev Board:

A single-board computer that is capable of performing high level Machine learning tasks using the Edge TPU coprocessor. Provides very fast inferencing for the ML models that is ideal for prototyping [5]. Uses Mendel Linux operating system.

B. Edge TPU technology:

Google's purpose-built accelerator application specific integrated circuit (ASIC) designed to run Artificial intelligence applications [2]. Low physical and power footprint while providing high accuracy and high performance.

C. TensorFlow Lite:

Light version of Tensor-flow that helps achieves low latency inference. These TFlite models can be made even smaller through quantization, converting 32-bit model to 8-bit model representations that is required by the Edge TPU [3]. Tensorflow lite models cannot be trained directly, a Tensorflow converter is required to convert Tensorflow file to Tensorflow Lite.

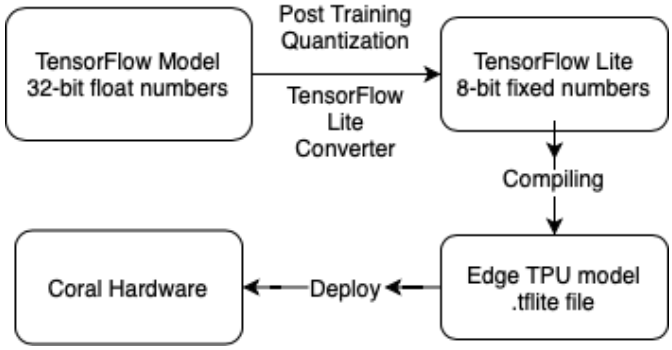


Fig. 1: The basic workflow to create a model for the Edge TPU

D. Benefits of on-device Machine learning

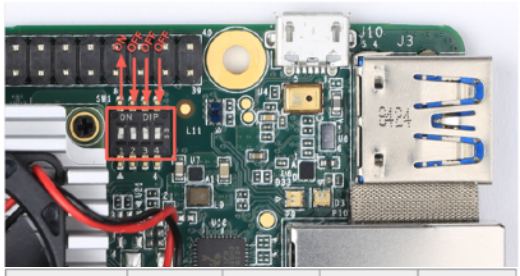
1. High performance as local Machine learning acceleration.
2. Better privacy: Data can stay on device and accessed when required.
3. Works offline: No internet required to connect to the board.
4. Power efficient: helps save bandwidth and power.
5. Local data accessible: sensor data is available on device.

E. Google Colab

Google Colab notebook is employed to mix executable code into a single document. It allows us to mount the google drive folder containing the pictures of the cars and the parking lot. Training and conversion of model to 8-bit is done using Google Colab notebook.

F. Configuration

Before starting the flashing process, make sure the board is not powered on and the boot mode switches are set to eMMC mode as shown in the Fig.2. Start off by connecting the host computer to the serial console on the dev board using a micro-B USB cable. The orange and green lights should illuminate on the board after you have successfully connected the board to the host computer. Connect the USB-C cable to the Data port on the coral board and start the flashing process.



Boot mode	Switch 1	Switch 2	Switch 3	Switch 4
eMMC	ON	OFF	OFF	OFF

Fig.2 :eMCC Boot Mode

G. Image Dataset

6000 Images of our model were taken with and without the cars through the Coral Pi camera. CNRPark Dataset was used for validation purposes. Pictures of parking lot were obtained during different times of the day with different perspectives and angles of view in order to get various light conditions. CNRPark contains 12,584 labelled patches and it is available for download. CNRPark parking spaces masks are non-rotated squares and often images do not cover precisely or entirely the parking space volume.

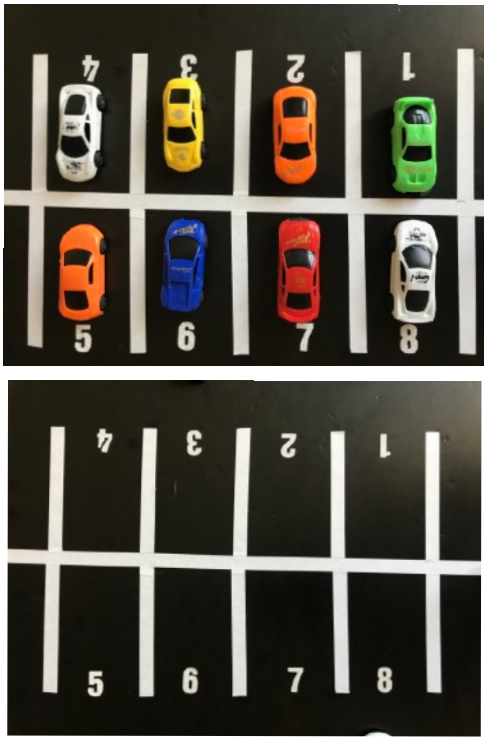


Fig.3 :Images captured using Coral pi camera

IV. METHODOLOGY

A. Parking detection model

Start off by creating a small parking model using mini cars, cardboard, and white tape to mark the parkings. All the parking spaces need to be numbered so it becomes easier for the model to identify which parking space has the car parked. Cars can be placed in any order anywhere.

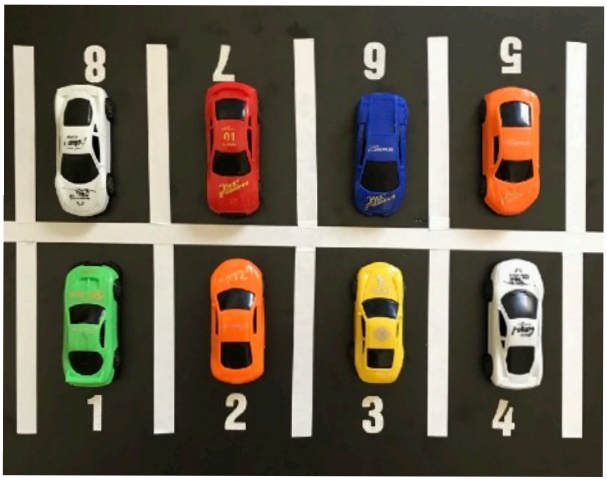


Fig.4 Parking model with cars parked

B. Clicking images for training

650 images were taken from different angles, with different cars in order to increase the model accuracy. Images were cropped, separated into 2 classes (i.e empty and parked spaces). Then the google drive folder containing the parking lot dataset was mounted on to Google colab.Then load and split the data into Training and validation. Then the images were augmented that included image flipping, random cropping, adding noise, and changing contrast. After this process a total of 3919 images were obtained. Then training set is displayed from the dataset.

```
Found 3919 files belonging to 2 classes.
Using 3136 files for training.
Found 3919 files belonging to 2 classes.
Using 783 files for validation.

Load Testing Data
Found 16 files belonging to 2 classes.
```

Fig. 5: Splitting dataset into training and validation

C. Defining a Sequential Model

Learning and understanding the complex patterns of the model with the help of activation functions like ReLU and sigmoid. The main purpose of using activation functions is to generate non-linearity into the output of neuron. One input tensor and one output tensor was used for each plain layer of the stack. List of layers were passed through the sequential layer constructors. All the Keras layers need to know the input shape in order to generate the weights. Total parameters utilized were 24,768,385 and the non-trainable parameters were 23,587,712 for this sequential model. Model.summary() was used to display the content of the block as shown in Figure 3.3.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 2048)              23587712
_________________________________________________________________
dense_3 (Dense)              (None, 512)               1049088
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 257
=================================================================
Total params: 24,768,385
Trainable params: 1,180,673
Non-trainable params: 23,587,712
_________________________________________________________________
```

Fig.6. Sequential model

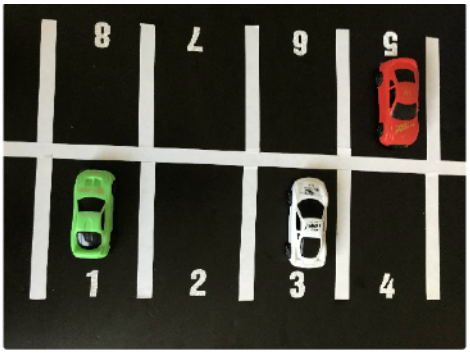
D. Transfer learning

With the help of transfer learning a Resnet model was defined. Resnet model contained a huge dataset with over 23 million trainable parameters. Base layer also known as the model layer was changed, layer appending was done to the final layer of the Resnet model. The last two layers were changed to the number of classes, which in our case were two i.e parked and empty spaces.

Model was first trained on the Resnet model but it gave lower detection rate and accuracy due to its low inference and high processing rate. MobileNet model contained 100,000 dataset images due to which we got a fast inference and detection rate. An inference of 13.5ms was achieved by the MobileNet model. The first inference on Edge TPU is slow because it includes loading the model into the TPU memory. Every parking space had to be mapped separately and that is the reason for the inference being 13.5ms.

E. Web app interface

Python module named Flask was used to create the Web app interface for our parking detection model. Tools and libraries required to build a Web app interface were provided for the Flask module. HTML and CSS scripts were used for website building and designing the display on the browser. The boundaries around the cars were defined using MSPaint. Coordinates were then used to map the cascades for the model. Port 8087 was used as the listening port for the browser.



Empty Parking Spaces

Full Parking Spaces

```
    "slots":{
        "1": {"top": [260, 850], "bottom": [750, 1630]},
        "2": {"top": [841, 881], "bottom": [1330, 1650]},
        "3": {"top": [1417, 905], "bottom": [1857, 1649]},
        "4": {"top": [1920, 930], "bottom": [2370, 1670]},
        "5": {"top": [1937, 121], "bottom": [2377, 881]},
        "6": {"top": [1409, 121], "bottom": [1850, 860]},
        "7": {"top": [849, 121], "bottom": [1353, 829]},
        "8": {"top": [281, 113], "bottom": [753, 809]}
    },
    "image_size": [2582, 1933],
    "display_grid": true
```

Fig.7 Mapping of Parking spaces

F. Executing the MobileNet model

Activate the Web app interface listening at port 8087 by running the main.py file using command "python main.py". Open the browser and type the Coral Dev board's IP address followed by the port number <IP:Port number> e.g- 192.168.100.2:8087. Coral Pi camera is held on top of the model, live video is captured, and the inference is done using the Coral Dev board. The output is shown on the web browser as seen in Fig. 11.

Values for the Empty and Full parking spaces are updated every 0.3 millisecond. When a vehicle parks in the parking spot, the color of the box changes from green to red indicating the parking is occupied and the value of Empty and Full parking space is updated.

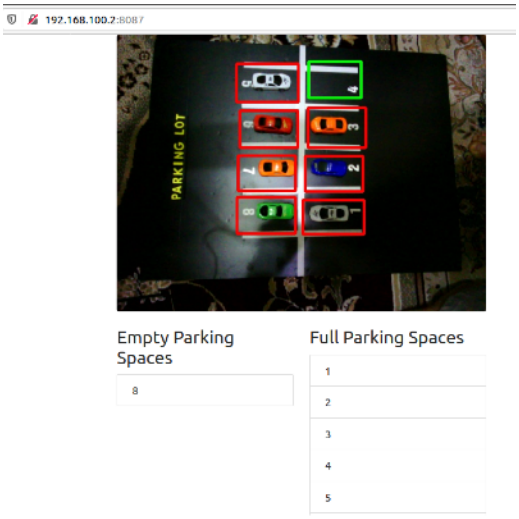


Fig.8 Model output

IV. Results And Discussion

A: Model Fitting

The model was fit to the data and the accuracy of the fit was analyzed. Accurate results were generated after fitting the model to data. Model was tested four times and the difference between the observed and predicted values was small and unbiased. An accuracy of 99.87% was achieved in this process. Before converting to TFlite model, the accuracy of the model with validation data and testing data was 99.6% and 100% respectively.

B: Evaluating Model accuracy after Model Quantization

Using the TFlite converter, the Tensorflow model was converted to the 8-bit quantized TFlite model. 32-bit images were converted to 8-bit images to make it compatible with the TFlite model. After converting the model to 8-bit TFlite model, the new model was uploaded on to the coral board using the mdt pull command.

Images were tested in batches. Accuracy test on the first batch of 16 images was 100%. Changing the batch count value to zero would test the accuracy of the quantized model with all the images. This model maintained the accuracy after the model quantization with one batch of images and took 2.8496 seconds to make inference. Therefore the final model accuracy achieved by our model was 100%.

```
    for image in val_images:
      prediction = classify_image(interpreter, image)
      prediction_values.append(prediction)
  batchProgress.value += 1

tflite_accuracy = tf.keras.metrics.Accuracy()
tflite_accuracy(prediction_values, truth_labels)
print("Quant TF Lite accuracy: {:.3%}".format(tflite_accuracy.result()))
```

```
Running accuracy test with 1 batches of size 16 images
Quant TF Lite accuracy: 100.000%
```

Fig.12 Accuracy post model quantization

V. Conclusions

A small-sized prototype of a modern parking lot was created in this project using toy cars, cardboard, and white tape. Pictures of the individual parking spaces were taken using the coral pi camera. Images were then flipped and random cropped as part of image augmentation to fit our model. This dataset was successfully trained using Transfer learning to identify the cars used in our prototype. The Web app interface created using the python module Flask, HTML, and CSS helped in achieving the right outputs. The availability of parking spaces in our parking lot prototype were updated every 0.3ms with the help of our Javascript code. As predicted, the color of the parking space, as detected by our model, changed from green to red when the car was parked. MobileNet model was tested using images of 20 different toy cars of roughly same dimensions. This test showed an accuracy of 100% in correctly detecting these 20 toy cars which indicates that our model can work successfully with real cars in parking lots outdoors.

VI. Future Scope

Nowadays, the numbers of cars are increasing and it is very hard to keep track of so many cars in a parking lot. Parking detection systems have been put in place in settings like malls to tackle this problem. However, these are inefficient in many ways . For instance, they do not detect the cars already parked in the parking space resulting in the parking detectors giving incorrect information to the user wanting to park their vehicle.

Future scope includes adding number plate detection system along with the our existing parking detection model in order to implement it in malls and private parking lots. This would help in tracking the time the car enters and exits the parking lot. Along with this, having cloud based entries would help track the duration the car is parked. This in turn reduces the need for printing parking tickets.

Additionally, having an app functionality would allow users to book the parking spaces using app. Offline mapping and guidance to the available parking spot can be added to improve functionality. Discount vouchers and benefits for app users could incentivize the use of our application which in turn would make it more appealing to potential investors.

Another possibility for our future work would be adding sensors, like LiDAR or IR sensor, to the parking spaces to compensate for the parking spaces that fall in the blind spot of the cameras in the malls. Having sensors for these parking lots can add up to the functionality of our model.

VI. References

1. Get started with the Dev Board. (n.d.). Retrieved December 19, 2020, from https://coral.ai/docs/dev-board/get-started

2. Cloud Tensor Processing Units (TPUs) | Google Cloud. (n.d.). Retrieved December 19, 2020, from https://cloud.google.com/tpu/docs/tpus

3. TensorFlow Lite guide. (n.d.). Retrieved December 19, 2020, from https://www.tensorflow.org/lite/guide

4. Artificial. (2020, February 17). 1. What is the Edge TPU? Retrieved December 20, 2020, from https://medium.com/@deve321/1-what-is-the-edge-tpu-a7dc7c5272dd

5. Imran, Hamza & Mujahid, Usama & Wazir, Saad & Latif, Usama & Mehmood, Kiran. (2020). Embedded Development Boards for Edge-AI: A Comprehensive Report.

6. Allan, A. (2019, April 25). Hands on with the Coral Dev Board. Retrieved December 20, 2020, from https://medium.com/@aallan/hands-on-with-the-coral-dev-board-adbcc317b6af

7. B. Pechiammal and J. A. Renjith, “An efficient approach for automatic license plate recognition system,” ICONSTEM 2017 - Proc. 3rd IEEE International Conference on Science Technology Engineering Management vol. 2018–January, pp. 121–129, 2018.

8. P.Ajitha, A. Sivasangari, K.Indira,” Predictive Inter and Intra Parking System”, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume- 8, Issue-2S, December 2018.

9. Novais, P.; Lloret, J.; Chamoso, P.; Carneiro, D.; Navarro, E.; Omatu, S.; Valdeolmillos, D.; Mezquita, Y.; Ludeiro, A.R. Sensing as a Service: An Architecture Proposal for Big Data Environments in Smart Cities. Adv. Intell. Syst. Comput. 2019, 97–104.

10. Paidi, V.; Fleyeh, H.; Hakansson, J.; Nyberg, R.G. Smart Parking Sensors, Technologies and Applications for Open Parking Lots: A Review. IET Intel. Transport Syst. 2018, 12, 735–741.

11. A. Soni and A. P. Singh, "Automatic Motorcyclist Helmet Rule Violation Detection using Tensorflow & Keras in OpenCV," *2020 IEEE International Students' Conference on Electrical,Electronics and Computer Science (SCEECS)*, Bhopal, India, 2020, pp. 1-5, doi: 10.1109/SCEECS48394.2020.55.

12. S. Chen, Y. Wei, Z. Xu, P. Sun and C. Wen, "Design and Implementation of Second-generation ID Card Number Identification Model based on TensorFlow," 2020 IEEE International Conference on Information Technology,Big Data and Artificial Intelligence (ICIBA), Chongqing, China, 2020, pp. 1146-1149, doi: 10.1109/ICIBA50161.2020.9276985.

13. Shadi A. Noghabi, Landon Cox, Sharad Agarwal, and Ganesh Ananthanarayanan. 2020. The Emerging Landscape of Edge Computing. GetMobile: Mobile Comp. and Comm. 23, 4 (December 2019), 11-20.

14. Emily Caveness, Paul Suganthan G. C., Zhuo Peng, Neoklis Polyzotis, Sudip Roy, and Martin Zinkevich. 2020. TensorFlow Data Validation: Data Analysis and Validation in Continuous ML Pipelines. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 2793–2796.

15. Stephen Guynes, James Parrish, and Richard Vedder. 2020. Edge computing societal privacy and security issues. <i>SIGCAS Comput. Soc. 48, 3–4 (September 2019), 11–12.