Table of Content

Abstract:

DeepPiCar uses Neural Network with Dave Architecture to run on a track autonomously. However, the model is susceptible to light intensity, reversing the car direction, and other environmental changes. The goal of the project is to fine-tune the already-trained model and make it adapt to these changes, instantly. This is done by retraining a part of the neural network(on-device) rather than training the whole network.

Introduction/Specification/Setup:

The embedded system used is the Raspberry pi 0 w 2. The specs are 512 KB SDRAM and 1GHz quad-core 64-bit Arm Cortex-A53 CPU. Please make sure Raspberry OS is 64-bit, otherwise, TensorFlow cannot be downloaded on a 32-bit as it is incompatible. The framework or the library used is TensorFlow for Deep Learning for fine-tuning the model on the new dataset. The original deep learning network on which the model was trained is DAVE Architecture.

For testing purposes, we will use the bright light dataset for training the base model. Then fine tune this model on a new dataset which is collected in dim lighting conditions.

Method:

Pull the DeepPiCar repository from https://github.com/heechul/DeepPicar-v3 and set up the workspace as directed in the Readme file provided. Create a virtual environment using python vitualvenv. This is done for setting up TensorFlow. Still, a normal pip install tendorflow may cause an error. From the https://bitsy.ai/3-ways-to-install-tensorflow-on-raspberry-pi/,

```
pip install --no-cache-dir tensorflow
```

Then upgrade the packages in pip from https://www.activestate.com/resources/quick-reads/how-to-update-all-python-packages/:

```
pip3 list --outdated --format=freeze | grep -v '^\-e' | cut -d = -f 1 | xargs -n1 pip3 install –U
```

There is an error that may pop up, once the fine-tuning is done and when tflite inference model is being generated:

Tflite model-maker in Colab TypeError: EndVector() missing 1 required positional argument: 'vectorNumElems'

This is due to the flatbuffer version installed along with TensorFlow will have a gibberish version like:

Flatbuffer v20434354356

However, Flatbuffer >=v2.0 is compatible with Tensorflow 2 and so install it in the following way:

pip install flatbuffers==2.0

The next step is to increase the size of the Swap File because the default size of the Swap File and RAM was not enough to train a part of the network. Please refer https://nebl.io/neblio-university/enabling-increasing-raspberry-pi-swap/ . The program kills itself if it runs out of memory. Moreover, disable GPU, which allows more RAM available for model training which can be done by

sudo raspi-config

Use the RullAll.ipynb to create a base model using the bright light dataset on Google Collab and save it in Pi zero. Next is to collect a new dataset and save it in new_dataset folder. Run *fine_tune.py* in the virtual environment created earlier on the device, and the file is present on the following link:

Link: https://drive.google.com/file/d/18SY3COjxpSV-lOfYLCJqTzku13m5FUW3/view?usp=sharing

The output for the above generates a model file and tflite inference file saves it into the models directory.

(Note: Before training on device, generate your fine-tune model in Google Collab using the following Python Notebook and test it practically:

Link: https://colab.research.google.com/drive/1j1BnmQxH3tDbark_CGSv4CPBr0YZzScO?usp=sharing

Then move on to on-device training.)

Analysis:

The second layer from the top and the second last layer from the bottom of the DAVE Architecture was found to be the two best layers to be fine-tuned for the new dataset. These layers were selected based on trial-and-errors, an ad hoc process. The rest of the layers were frozen in the following way:

```
model.trainable = True
for layer in model.layers[:]:
layer.trainable = False
    model.layers[2].trainable = True
    model.layers[14].trainable = True
```

The reason for the above way of writing code is that I suspect there is a bug in TensorFlow. Let's if model layers are frozen in the following way:

```
model.trainable = False
        model.layers[2].trainable = True
        model.layers[14].trainable = True
```

If you print model.summary(), the total number of trainable params will remain zero. However, in the former case the number of trainable params = 2,334, even though both pieces of code have the same logical meaning.

For the base model, the learning rate was 5e-5. For fine-tuning increase It to 3e-4, to make the training faster and set the epoch as 5. The local optima are reached within these 5 epochs. For 737 samples of the newly collected data, it took **approximately 1 to 2 hours** of on-device training.

Certain factors to keep in mind is checking the overall accuracy is not ideal feedback for the new model generate. Rather keep an eye on the F1 scores of each class (center, left, right). Moreover, testing all models generated practically will make you understand if the model was overfitted or underfitted and according to change hyperparameters if needed.

Conclusion:

This project proves that it's possible to run TensorFlow on Raspberry Pi 0 2w and fine-tuning helps the model to adapt to new changes in the environment. Moreover, a new small dataset collected can fine-tune the based model in an hour or two. These most significant two layers were found through excitement and practical testing of the model.

Future Work:

Instead of a trial-and-error approach, proper mathematical reasoning is to be assessed. Additionally, a dedicated button on the joystick can be chosen for one-click training. Possibly an 8-bit LCD be added but it would increase the cost. Instead, adding one LED would be a great option. If it glows it can indicate the model is in the training process and vice versa.

To test if these 2 layers still hold as the best layers for fine-tuning, test out fine_tune.py by collecting new data on a new track or just by reversing the direction of the car on the same track.