TensorFlow Hand Gesture Recognition by Raspberry-Pi

Team Name:- *Rivets*

Team Members:- Rijul Jain, Pushpendra Kumar, Shivam Kumar, Harshit Krishna, Pratik Kamble

Things Used in this Project:-

Software Components: TensorFlow, Fritzing.

Main Crux behind the Project:-

The idea behind this project is to create a device able to drive an actuator (in which 7 flags with different colors, made up of polymorphism stands, after recognizing the class corresponding to each flag) based on the gesture of the hand's fingers. The project is specialized in recognizing streaming images of the hand taken by the raspberry-pi camera. The data set of the images used to train the model was created ad hoc with images taken from the Raspberry Camera only (not other devices) with a neutral background.

The model training and testing were performed and the same can be viewed using this link.

Once these steps were concluded, the final result now has to be moved to the Raspberry Pi. The Raspberry Pi is supposed to perform only the inference from the image streaming taken from the Raspberry Pi camera (which is computational so, much less intensive than the training).

```
export_dir = "E:\Downloads\MLSessions-main\Model_Hand_Gestures"

loaded = tf.saved_model.load(export_dir)

converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)

converter.optimizations = [tf.lite.Optimize.DEFAULT]

tflite_model = converter.convert()

tflite_model_file = 'Model_Hand_Gestures.tflite'
```

```
with open(tflite_model_file, "wb") as f:

    f.write(tflite_model)
```

Following the code, the model is saved in a file ("enzo_02"), which can easily be copied from the desktop to the Raspberry PI filesystem. Then, it can be read by a python script running on the Raspberry Pi.

```
from tflite_runtime.interpreter import Interpreter

....

interpreter = Interpreter(args.model)

 interpreter.allocate_tensors()

 _, height, width, _ = interpreter.get_input_details()[0]['shape']
```

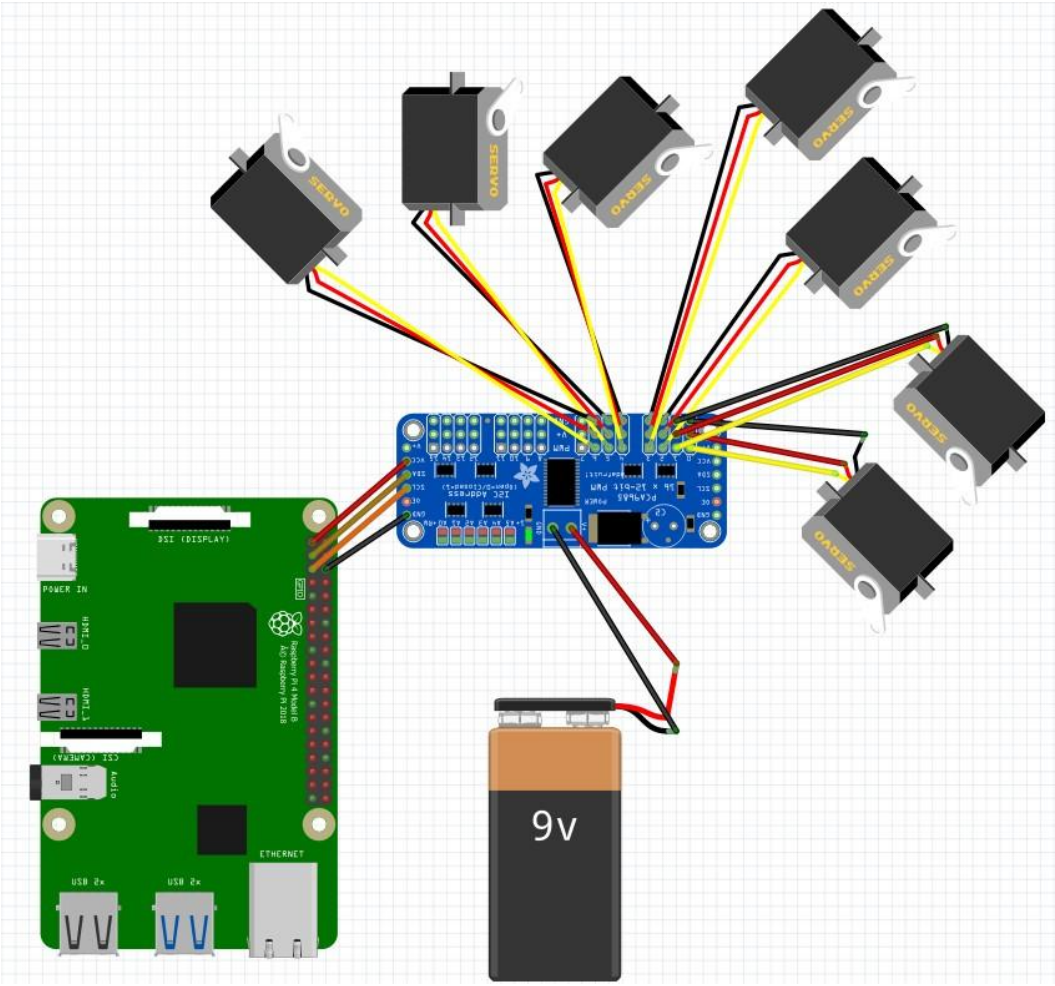
The full code is already given in the above link.
Below is an example of how to launch the script:-

```
python3 classify_picamera_servo.py   --model
./Model_Hand_Gestures.tflite   --labels ./labels
```

Electric Circuits Overview:-

The electric circuit is quite straightforward. The Raspberry Pi 4 with the camera are the core components.

They collect the video stream and perform the inference using the deployed model. Then, based on the inference results, a signal is sent to the PCA9685 which actions the 7 servos accordingly. The PCA9685 workload to signal the 7 servos is supported by an external battery of 9V.



Tasks Automated:-

The hand gestures have been divided into 7 classes-

1. Palm
2. ‘L’ Shape
3. Fist
4. Thumbs Up
5. Index Finger
6. Gesture ‘OK’
7. ‘C’ Shape

The model recognizes the class using the input image taken by the camera. According to the class predicted by the model, certain tasks have been automated. These tasks are as follows:

Gestures	**Tasks Automated**
Palm ✋	Launch the File Explorer
'L' Shape 🤙	Show the Help Menu
Fist 👊	Capture a Screenshot
Thumbs Up 👍	Start Voice Recording
Index Finger 👆	Volume Up
Gesture 'OK' 👌	Play Music
'C' Shape 🫲	Volume Down

Code Running on the Raspberry-Pi:-

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import io
import time
import numpy as np
import picamera

from time import sleep
```

```
from adafruit_servokit import ServoKit

from PIL import Image
from tflite_runtime.interpreter import Interpreter

kit = ServoKit(channels=16)


def load_labels(path):
    with open(path, 'r') as f:
        return {i: line.strip() for i, line in enumerate(f.readlines())}


def set_input_tensor(interpreter, image):
    tensor_index = interpreter.get_input_details()[0]['index']
    input_tensor = interpreter.tensor(tensor_index)()[0]
    input_tensor[:, :] = image


def classify_image(interpreter, image, top_k=1):
    """Returns a sorted array of classification results."""
    set_input_tensor(interpreter, image)
    interpreter.invoke()
    output_details = interpreter.get_output_details()[0]
    #print(output_details['index'])
    #print(interpreter.get_tensor(output_details['index']))
    output = np.squeeze(interpreter.get_tensor(output_details['index']))
    print(output)
    # If the model is quantized (uint8 data), then dequantize the results
    if output_details['dtype'] == np.uint8:
        scale, zero_point = output_details['quantization']
        output = scale * (output - zero_point)

    ordered = np.argpartition(-output, top_k)
    print(ordered[0])
    return [(i, output[i]) for i in ordered[:top_k]]


def servo_ctrl(id=0):
    for i in range(7):
```

```
        if i == id:
            kit.servo[i].angle = 90
        else:
            kit.servo[i].angle = 10
    return 0


def main():

    parser = argparse.ArgumentParser(
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--model',
                        help='File path of .tflite file.',
                        required=True)
    parser.add_argument('--labels',
                        help='File path of labels file.',
                        required=True)
    args = parser.parse_args()

    labels = load_labels(args.labels)

    interpreter = Interpreter(args.model)
    interpreter.allocate_tensors()
    _, height, width, _ = interpreter.get_input_details()[0]['shape']

    with picamera.PiCamera(resolution=(640, 480),
                           framerate=30) as camera:  # it was 30
        camera.start_preview()
        try:
            stream = io.BytesIO()
            for _ in camera.capture_continuous(stream,
                                               format='jpeg',
                                               use_video_port=True):
                stream.seek(0)

                img = Image.open(stream).convert('RGB').resize((width,
                                         height),Image.ANTIALIAS)
                img = np.array(img, dtype=np.float32)
                img = img / 255.
```

```
                # Add a batch dimension
                input_data = np.expand_dims(img, axis=0)

                start_time = time.time()
                results = classify_image(interpreter, img)
                elapsed_ms = (time.time() - start_time) * 1000
                label_id, prob = results[0]
                stream.seek(0)
                stream.truncate()
                camera.annotate_text = '%s %.2f\n%.1fms' %
                                (labels[label_id], prob, elapsed_ms)

                servo_ctrl(id=label_id)
                #time.sleep(5) #Enzo
        finally:
            camera.stop_preview()


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

Hand Gestures Recognition and Automation (Perceptron and CNN)

Goal is to recognize hand gestures. We've trained the model on our own dataset using *Perceptron* (and *CNN* thereafter). We've included our dataset in the repository itself. In it's present state the model is trained to recognize 7 gestures but can easily be trained for many hand gestures. We have also uploaded the code that we used for capturing the hand and processing it for training the model.

Model gives a high train accuracy: *99.95%* and validation accuracy: *98.81%*

Images in the dataset are of dimension *200 by 200*. But for performance reasons they have been resized to *50 by 50*.

What's in the Repository

- `captureHandGesture.py` - This program can capture new hand gestures and write them in the specified directory
- `model_genetor.ipynb` - This program uses the given dataset to train the Perceptron model and develops the model.
- `classify_picamera_servo.py` - This program is for Raspberry.
- `Prediction_Laptop_Gestures_Automation.ipynb` - This program uses the saved model to classify and automate classes.

Sample of Images from the Dataset

- C

- Fist

- Index

- L

- OK

- Palm

- Thumb

Model Outputs

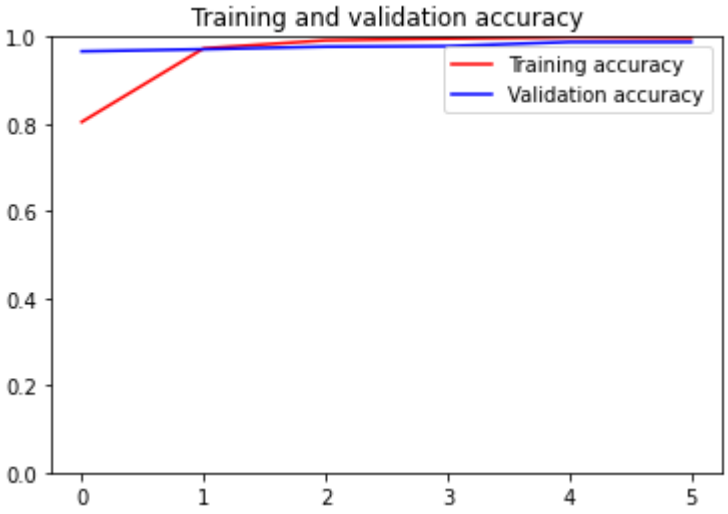
- Model Summary

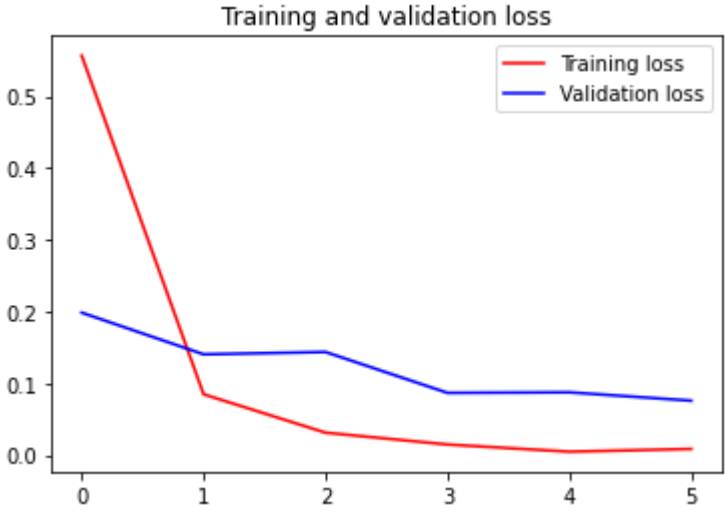
```
Model: "CIFAR_Sequential_CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 33856)             0
_________________________________________________________________
dense (Dense)                (None, 256)               8667392
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 903
=================================================================
Total params: 8,720,007
Trainable params: 8,720,007
Non-trainable params: 0
_________________________________________________________________
```

- Training Output

```
Epoch 1/6
31/31 [==============================] - 36s 139ms/step - loss: 0.5570 - accuracy: 0.8046 - val_loss: 0.1985 - val_accuracy: 0.9667
Epoch 2/6
31/31 [==============================] - 4s 127ms/step - loss: 0.0845 - accuracy: 0.9735 - val_loss: 0.1403 - val_accuracy: 0.9714
Epoch 3/6
31/31 [==============================] - 4s 125ms/step - loss: 0.0310 - accuracy: 0.9918 - val_loss: 0.1435 - val_accuracy: 0.9774
Epoch 4/6
31/31 [==============================] - 4s 126ms/step - loss: 0.0145 - accuracy: 0.9959 - val_loss: 0.0865 - val_accuracy: 0.9786
Epoch 5/6
31/31 [==============================] - 4s 125ms/step - loss: 0.0045 - accuracy: 0.9990 - val_loss: 0.0873 - val_accuracy: 0.9881
Epoch 6/6
31/31 [==============================] - 4s 124ms/step - loss: 0.0083 - accuracy: 0.9974 - val_loss: 0.0756 - val_accuracy: 0.9881
```

- Graphs


Training and validation accuracy
Training accuracy
Validation accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5


Training and validation loss
Training loss
Validation loss
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
5

Vision Arcadia

Rivets presents,

Hand Gestures Automation System

1. Hand Gesture (c) - **Volume Down**

2. Hand Gesture (Fist) - **Capture Screenshot**

3. Hand Gesture (Index Finger) - **Volume Up**

4. Hand Gesture (L) - **Open Help.**

5. Hand Gesture (OK) - **Play Music.**

6. Hand Gesture (Palm) - **Open File Explorer.**

7. Hand Gesture (Thumbs Up) - **Start Voice Recording.**