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']
```

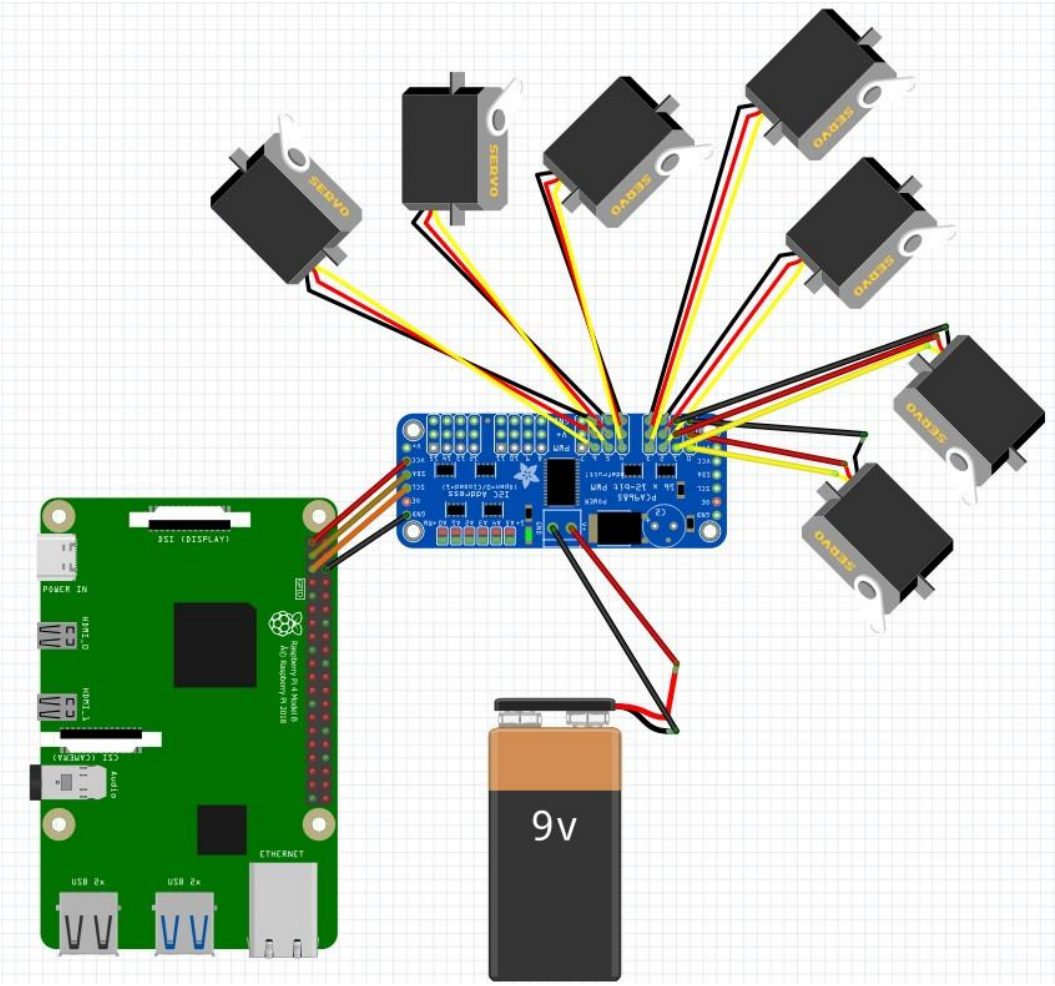
The full code is already given in the above link.
Below is an example of how to launch the script:-

```
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):
```

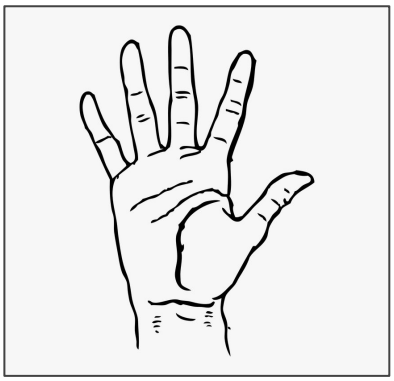
```
        if i == id:
            kit.servo[i].angle = 90
        else:
            kit.servo[i].angle = 10
    return 0


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()
```

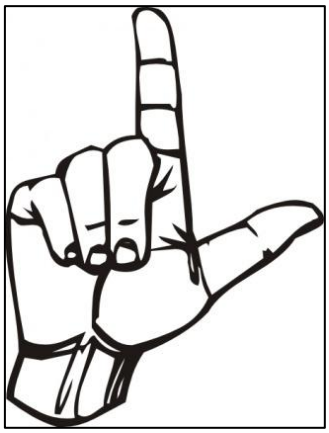
Vision Arcadia

Rivets presents,

Hand Gestures Automation System

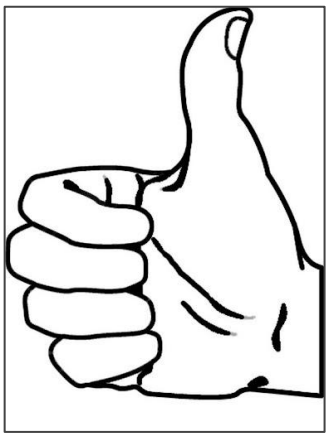


1. Hand Gesture (Palm) -
   Open File Explorer.



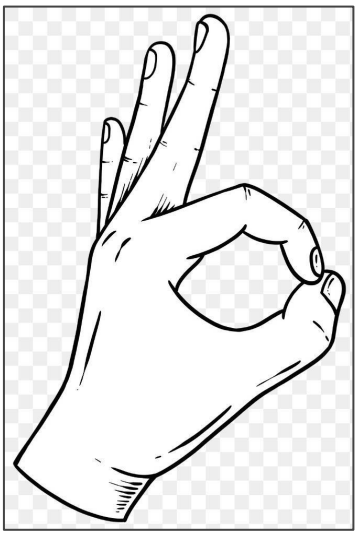
2. Hand Gesture (L) -
   Open Help.

3. Hand Gesture (Fist) -
   Capture Screenshot



4. Hand Gesture (Thumbs Up) -
   Start Voice Recording.

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



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

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