CE4172: Internet of Things – Tiny Machine Learning

Project Report

Tan Kah Heng Darryl

U1921321H

Introduction

Sign Language requires lots of effort to understand, especially when face to face and there is no available translator. Therefore, an Arduino Nano Sense Board with Deep Learning can be used to interpret sign language for those with no knowledge of sign language.

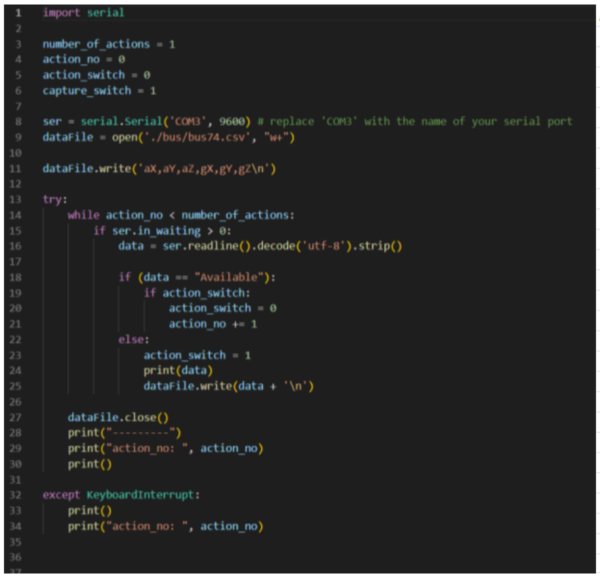
This project builds upon the Gesture Recognition conducted in lab, modifying the gesture detection to involve gestures from common phrases in Singapore Sign Language. These gestures are taken from Youtube.[1,2]

Data Collection

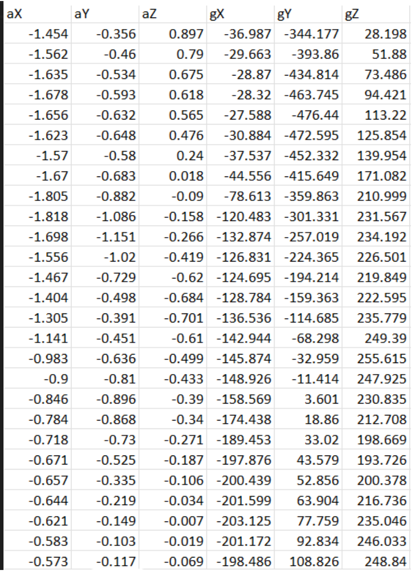
The gestures recorded from the sign language library are: ‘how’, ‘bus’, ’me’, ’you’. These words are selected as their gestures are distinct and do not feature similar movement to each other. These are important as explained in the problems faced in the project.

75 examples were recorded for each gesture and each gesture containing 119 samples per second from the IMU. Training, validation and test sets were split 60%-20%-20%.

While the Nano 33 BLE Sense Board prints the IMU output to the serial port, a python script is used to gather the information from the serial port and create a corresponding csv file to be kept for pre-processing and training later.



```python
import serial

number_of_actions = 1
action_no = 0
action_switch = 0
capture_switch = 1

ser = serial.Serial('COM3', 9600) # replace 'COM3' with the name of your serial port
dataFile = open('./bus/bus74.csv', "w+")

dataFile.write('aX,aY,aZ,gX,gY,gZ\n')

try:
    while action_no < number_of_actions:
        if ser.in_waiting > 0:
            data = ser.readline().decode('utf-8').strip()

            if (data == "Available"):
                if action_switch:
                    action_switch = 0
                    action_no += 1
            else:
                action_switch = 1
                print(data)
                dataFile.write(data + '\n')

    dataFile.close()
    print("---------")
    print("action_no: ", action_no)
    print()

except KeyboardInterrupt:
    print()
    print("action_no: ", action_no)
```



aX	aY	aZ	gX	gY	gZ
-1.454	-0.356	0.897	-36.987	-344.177	28.198
-1.562	-0.46	0.79	-29.663	-393.86	51.88
-1.635	-0.534	0.675	-28.87	-434.814	73.486
-1.678	-0.593	0.618	-28.32	-463.745	94.421
-1.656	-0.632	0.565	-27.588	-476.44	113.22
-1.623	-0.648	0.476	-30.884	-472.595	125.854
-1.57	-0.58	0.24	-37.537	-452.332	139.954
-1.67	-0.683	0.018	-44.556	-415.649	171.082
-1.805	-0.882	-0.09	-78.613	-359.863	210.999
-1.818	-1.086	-0.158	-120.483	-301.331	231.567
-1.698	-1.151	-0.266	-132.874	-257.019	234.192
-1.556	-1.02	-0.419	-126.831	-224.365	226.501
-1.467	-0.729	-0.62	-124.695	-194.214	219.849
-1.404	-0.498	-0.684	-128.784	-159.363	222.595
-1.305	-0.391	-0.701	-136.536	-114.685	235.779
-1.141	-0.451	-0.61	-142.944	-68.298	249.39
-0.983	-0.636	-0.499	-145.874	-32.959	255.615
-0.9	-0.81	-0.433	-148.926	-11.414	247.925
-0.846	-0.896	-0.39	-158.569	3.601	230.835
-0.784	-0.868	-0.34	-174.438	18.86	212.708
-0.718	-0.73	-0.271	-189.453	33.02	198.669
-0.671	-0.525	-0.187	-197.876	43.579	193.726
-0.657	-0.335	-0.106	-200.439	52.856	200.378
-0.644	-0.219	-0.034	-201.599	63.904	216.736
-0.621	-0.149	-0.007	-203.125	77.759	235.046
-0.583	-0.103	-0.019	-201.172	92.834	246.033
-0.573	-0.117	-0.069	-198.486	108.826	248.84

Figure 1 - Python code (left) used to listen to serial port and create csv file (right)

[1] https://www.youtube.com/watch?v=AT4pSTXxI5U
[2] https://www.youtube.com/watch?v=-PJOW098IKE

Model and Training Process

Using the data collected, the training was done using Google Colab. Below is the model that performs the best training and validation results.

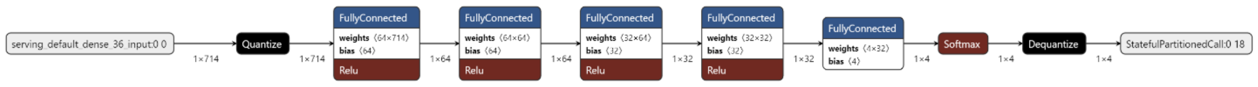


Figure 2 - Model in detail used for training

The model consists of

- 2 Dense layers with 64 neurons and ReLu activation function,
- 2 Dense layers with 32 neurons and ReLu activation function,
- 1 Dense layer with 4 neurons corresponding to the number of gestures and outputs a softmax.

Results

Below shows the results of the model after training:

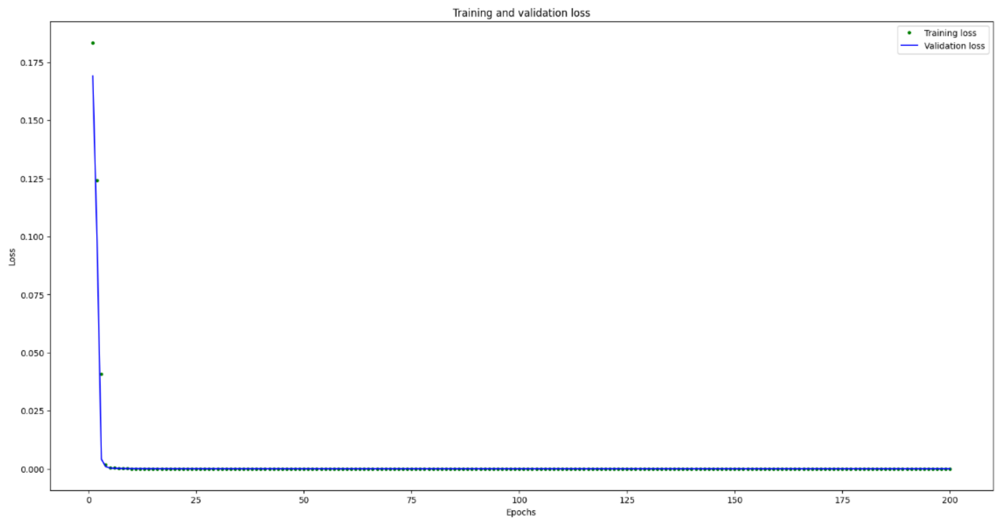


Figure 3 - Training and validation loss graph against epochs

Above shows the training and validation loss. The training and validation loss converges to close to 0 loss after a few epochs.

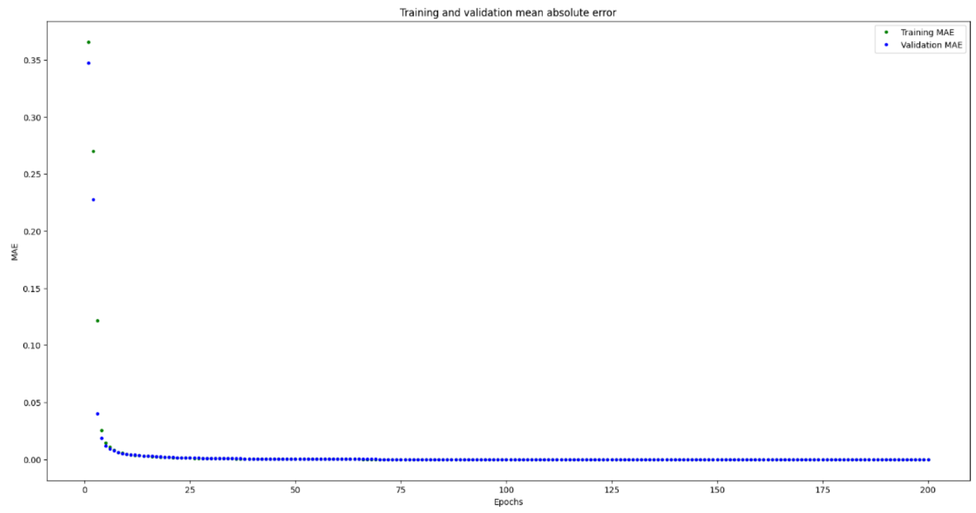
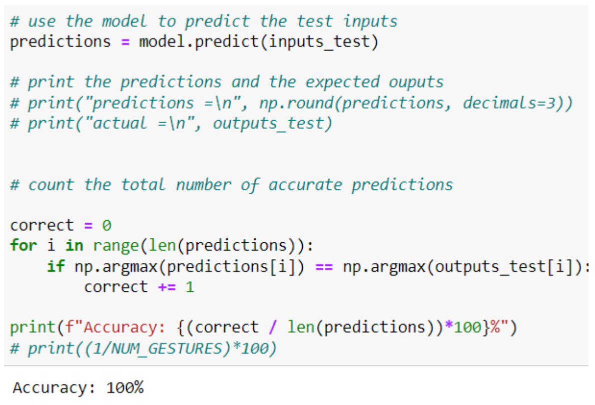


Figure 4 - Training and Validation Mean Absolute Error Graph against epochs

Above shows the mean absolute error for training and validation. The error also converges to 0 for both the training and validation data.
This shows that the model trained is very good and manage to perform very well on the training and validation set. Therefore, the model is able to distain the different gestures corresponding to the different sign language words.



```
# use the model to predict the test inputs
predictions = model.predict(inputs_test)

# print the predictions and the expected ouputs
# print("predictions =\n", np.round(predictions, decimals=3))
# print("actual =\n", outputs_test)


# count the total number of accurate predictions

correct = 0
for i in range(len(predictions)):
    if np.argmax(predictions[i]) == np.argmax(outputs_test[i]):
        correct += 1

print(f"Accuracy: {(correct / len(predictions))*100}%")
# print((1/NUM_GESTURES)*100)
```

```
Accuracy: 100%
```

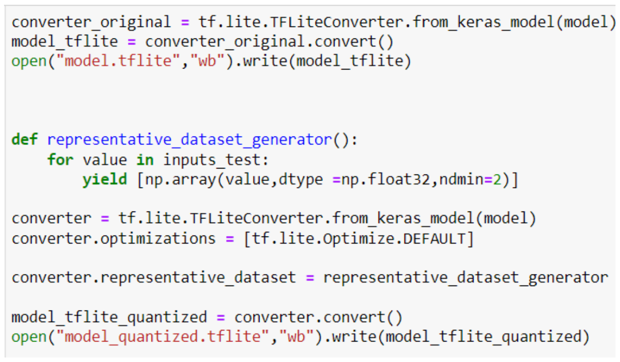
Figure 5 - Calculating Accuracy using test set

The test set is used to observe the performance of the model by calling model.predict() on the test set and comparing the predicted one-hot encoded output with the actual one-hot encoded output. The model gives an accuracy of 100%.

Optimization Techniques Used

1. Full integer quantization

Quantization is used to reduce the size of the model in order for the model to perform faster and at a lower power for the Nano 33 BLE Sense Board. The weights, biases and activation functions are quantized to 8 bits.



```
converter_original = tf.lite.TFLiteConverter.from_keras_model(model)
model_tflite = converter_original.convert()
open("model.tflite","wb").write(model_tflite)



def representative_dataset_generator():
    for value in inputs_test:
        yield [np.array(value,dtype =np.float32,ndmin=2)]

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]

converter.representative_dataset = representative_dataset_generator

model_tflite_quantized = converter.convert()
open("model_quantized.tflite","wb").write(model_tflite_quantized)
```

Figure 6 - Full integer quantization

After quantization, the model is reduced by 73.24% from 215,756 bytes to 57,728 bytes. The quantised model was tested on the test set once again and managed to sustain the performance of 100% accuracy.

	Size
Before quantization	215,756 bytes
After quantization	57,728 bytes
Difference	158,028 bytes

Problems faced during the training.

Sign Language often uses fingers to denote letters as well, which the IMU could not capture. Multiple words correspond to the same type of gesture. (e.g. Good and Thank you) Simple Deep NN could not detect subtle gesture differences using IMU (Accelerometer and gyroscopes can only record changes in movement).

Some improvement works that can be done would be:

1. Instead of using a normal deep learning, can explore using convolutional neural networks instead to recognise complex gestures.
2. Attach another IMU or Nano 33 BLE Sense Board on the other hand as sign language requires 2 hand gestures.
3. Find a way to record finger movements as well as most gestures requires finger movement.
4. Find a way to distinct 2 similar gestures.