****

(All rights reserved)

**Interns**: Evans Acheampong and Josiah Lansah

**Course of study**: Computer Engineering and Computer Science

# **Instructor:** Douglas T. Ayitey

# **Date:** October 28th, 2022

**DAILY INTERNSHIP REPORT**

**DAY 8**

**PRATICAL MACHINE LEARNING PROJECTS WITH ARDUINO NANO BLE 33 SENSE**

* **GESTURE RECOGNITION USING ARDUINO NANO 33 BLE SENSE (THIRD PHASE)**
* **ADDED EMOJI AND EXTRA GESTURES (FINAL PHASE)**

**Introduction**

Gesture recognition is an integral part of the Machine Learning world. Machines simply work with data that has been fed to it and trained by it. We explore further applications of gesture recognition using Arduino Nano 33 BLE Sense to add extra gestures for more experience. These include stab, slash, and upper.

**Goals and objectives**

* Adequately training data and including other gestures using Google Colab.
* Testing out the emoji buttons on the extra gestures.
* Exploring practical applications of the project such as remote physiotherapy in future projects.

**Hardware and Software Required**

* An [Arduino Nano 33 BLE Sense](https://store.arduino.cc/nano-33-ble-sense) board
* A Micro USB cable to connect the Arduino board to your desktop machine.
* To program the board, we used the [Arduino IDE](https://www.arduino.cc/en/main/software).
* Google Colab

**Procedure**

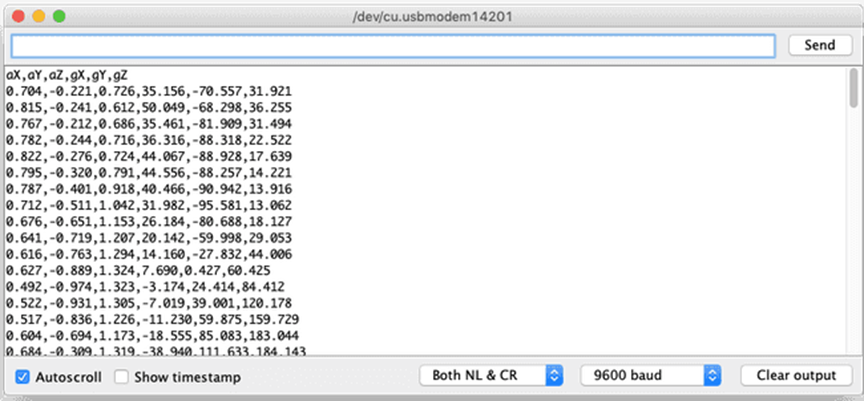
* Add some more gestures to go with the punch (👊) and flex
* Load the IMU\_Capture sketch on your Arduino Nano 33 BLE
* Caputure some additional gestures for a new emoji. Perhaps a 😄, 👍, 👏, or 👋
* Load the new CSV files into your Colab model
* Put the trained model back on your Nano 33 BLE
* Try increasing and decreasing the number of recordings per gesture
* Try to only use the accelerometer or gyroscope data (not both)
* Tweak the model structure and parameters

## **Capturing Gesture Training Data**

To capture data as a CSV log to upload to TensorFlow, we used **Arduino IDE > Tools > Serial Monitor** to view the data and export it to your desktop machine:

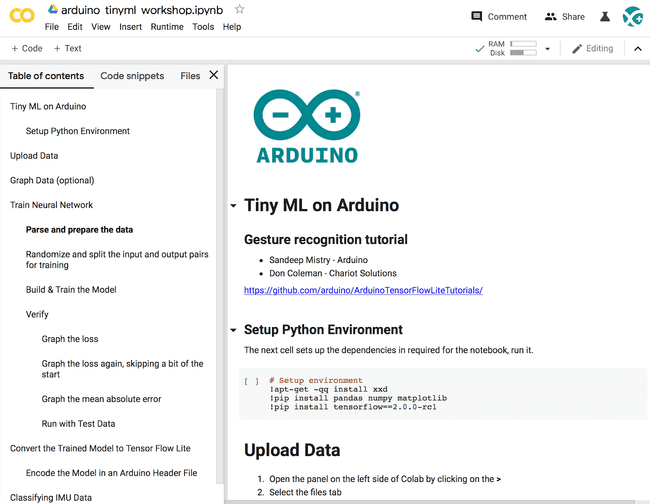
* Reset the board by pressing the small white button on the top
* Pick up the board in one hand (picking it up later will trigger sampling)
* In the Arduino IDE, open the Serial Monitor Tools > Serial Monitor
* If you get an error that the board is not available, reselect the port:
* Tools > Port > port name (Arduino Nano 33 BLE)
* Make a stab gesture with the board in your hand (Be careful whilst doing this!)
* Make the outward stab quickly enough to trigger the capture
* Return to a neutral position slowly so as not to trigger the capture again
* Repeat the gesture capture step 10 or more times to gather more data
* Copy and paste the data from the Serial Console to new text file called punch.csv
* Clear the console window output and repeat all the steps above, this time with the slash and upper gestures in files called slash.csv and stab.csv
* Make the inward slash and stab fast enough to trigger capture returning slowly each time.

**Note:** the first line of your three csv files should contain the fields aX,aY,aZ,gX,gY,gZ.



## Training in TensorFlow

We employed [Google Colab](https://colab.research.google.com/) to train our machine learning model using the data we collected from the Arduino board in the previous section. Colab provides a Jupyter notebook that allows us to run our TensorFlow training in a web browser.



**FULL CODE USED**

/\*

  IMU Classifier

  This example uses the on-board IMU to start reading acceleration and gyroscope

  data from on-board IMU, once enough samples are read, it then uses a

  TensorFlow Lite (Micro) model to try to classify the movement as a known gesture.

  Note: The direct use of C/C++ pointers, namespaces, and dynamic memory is generally

        discouraged in Arduino examples, and in the future the TensorFlowLite library

        might change to make the sketch simpler.

  The circuit:

  - Arduino Nano 33 BLE or Arduino Nano 33 BLE Sense board.

  Created by Don Coleman, Sandeep Mistry

  Modified by Dominic Pajak, Sandeep Mistry

  This example code is in the public domain.

\*/

#include <Arduino\_LSM9DS1.h>

#include <TensorFlowLite.h>

#include "tensorflow/lite/micro/kernels/micro\_ops.h"

#include <tensorflow/lite/micro/all\_ops\_resolver.h>

#include <tensorflow/lite/micro/micro\_error\_reporter.h>

#include <tensorflow/lite/micro/micro\_interpreter.h>

#include "tensorflow/lite/micro/micro\_mutable\_op\_resolver.h"

#include <tensorflow/lite/schema/schema\_generated.h>

#include <tensorflow/lite/version.h>

#include "model.h"

const float accelerationThreshold = 2.5; // threshold of significant in G's

const int numSamples = 119;

int samplesRead = numSamples;

// global variables used for TensorFlow Lite (Micro)

tflite::MicroErrorReporter tflErrorReporter;

// pull in all the TFLM ops, you can remove this line and

// only pull in the TFLM ops you need, if would like to reduce

// the compiled size of the sketch.

tflite::AllOpsResolver tflOpsResolver;

const tflite::Model\* tflModel = nullptr;

tflite::MicroInterpreter\* tflInterpreter = nullptr;

TfLiteTensor\* tflInputTensor = nullptr;

TfLiteTensor\* tflOutputTensor = nullptr;

// Create a static memory buffer for TFLM, the size may need to

// be adjusted based on the model you are using

constexpr int tensorArenaSize = 8 \* 1024;

byte tensorArena[tensorArenaSize] \_\_attribute\_\_((aligned(16)));

// array to map gesture index to a name

const char\* GESTURES[] = {

  "punch 👊👊👊👊👊",

  "slash 🤚🤚🤚🤚🤚",

  "stab 🔪🔪🔪🔪🔪",

  "upper 👐👐👐👐👐",

  "flex 💪💪💪💪💪"

};

#define NUM\_GESTURES (sizeof(GESTURES) / sizeof(GESTURES[0]))

void setup() {

  Serial.begin(9600);

  while (!Serial);

  // initialize the IMU

  if (!IMU.begin()) {

    Serial.println("Failed to initialize IMU!");

    while (1);

  }

  // print out the samples rates of the IMUs

  Serial.print("Accelerometer sample rate = ");

  Serial.print(IMU.accelerationSampleRate());

  Serial.println(" Hz");

  Serial.print("Gyroscope sample rate = ");

  Serial.print(IMU.gyroscopeSampleRate());

  Serial.println(" Hz");

  Serial.println();

  // get the TFL representation of the model byte array

  tflModel = tflite::GetModel(model);

  if (tflModel->version() != TFLITE\_SCHEMA\_VERSION) {

    Serial.println("Model schema mismatch!");

    while (1);

  }

  // Create an interpreter to run the model

  tflInterpreter = new tflite::MicroInterpreter(tflModel, tflOpsResolver, tensorArena, tensorArenaSize, &tflErrorReporter);

  // Allocate memory for the model's input and output tensors

  tflInterpreter->AllocateTensors();

  // Get pointers for the model's input and output tensors

  tflInputTensor = tflInterpreter->input(0);

  tflOutputTensor = tflInterpreter->output(0);

}

void loop() {

  float aX, aY, aZ, gX, gY, gZ;

  // wait for significant motion

  while (samplesRead == numSamples) {

    if (IMU.accelerationAvailable()) {

      // read the acceleration data

      IMU.readAcceleration(aX, aY, aZ);

      // sum up the absolutes

      float aSum = fabs(aX) + fabs(aY) + fabs(aZ);

      // check if it's above the threshold

      if (aSum >= accelerationThreshold) {

        // reset the sample read count

        samplesRead = 0;

        break;

      }

    }

  }

  // check if the all the required samples have been read since

  // the last time the significant motion was detected

  while (samplesRead < numSamples) {

    // check if new acceleration AND gyroscope data is available

    if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {

      // read the acceleration and gyroscope data

      IMU.readAcceleration(aX, aY, aZ);

      IMU.readGyroscope(gX, gY, gZ);

      // normalize the IMU data between 0 to 1 and store in the model's

      // input tensor

      tflInputTensor->data.f[samplesRead \* 6 + 0] = (aX + 4.0) / 8.0;

      tflInputTensor->data.f[samplesRead \* 6 + 1] = (aY + 4.0) / 8.0;

      tflInputTensor->data.f[samplesRead \* 6 + 2] = (aZ + 4.0) / 8.0;

      tflInputTensor->data.f[samplesRead \* 6 + 3] = (gX + 2000.0) / 4000.0;

      tflInputTensor->data.f[samplesRead \* 6 + 4] = (gY + 2000.0) / 4000.0;

      tflInputTensor->data.f[samplesRead \* 6 + 5] = (gZ + 2000.0) / 4000.0;

      samplesRead++;

      if (samplesRead == numSamples) {

        // Run inferencing

        TfLiteStatus invokeStatus = tflInterpreter->Invoke();

        if (invokeStatus != kTfLiteOk) {

          Serial.println("Invoke failed!");

          while (1);

          return;

        }

        // Loop through the output tensor values from the model

        for (int i = 0; i < NUM\_GESTURES; i++) {

          Serial.print(GESTURES[i]);

          Serial.print(":");

          Serial.println(tflOutputTensor->data.f[i], 6);

        }

        Serial.println();

      }

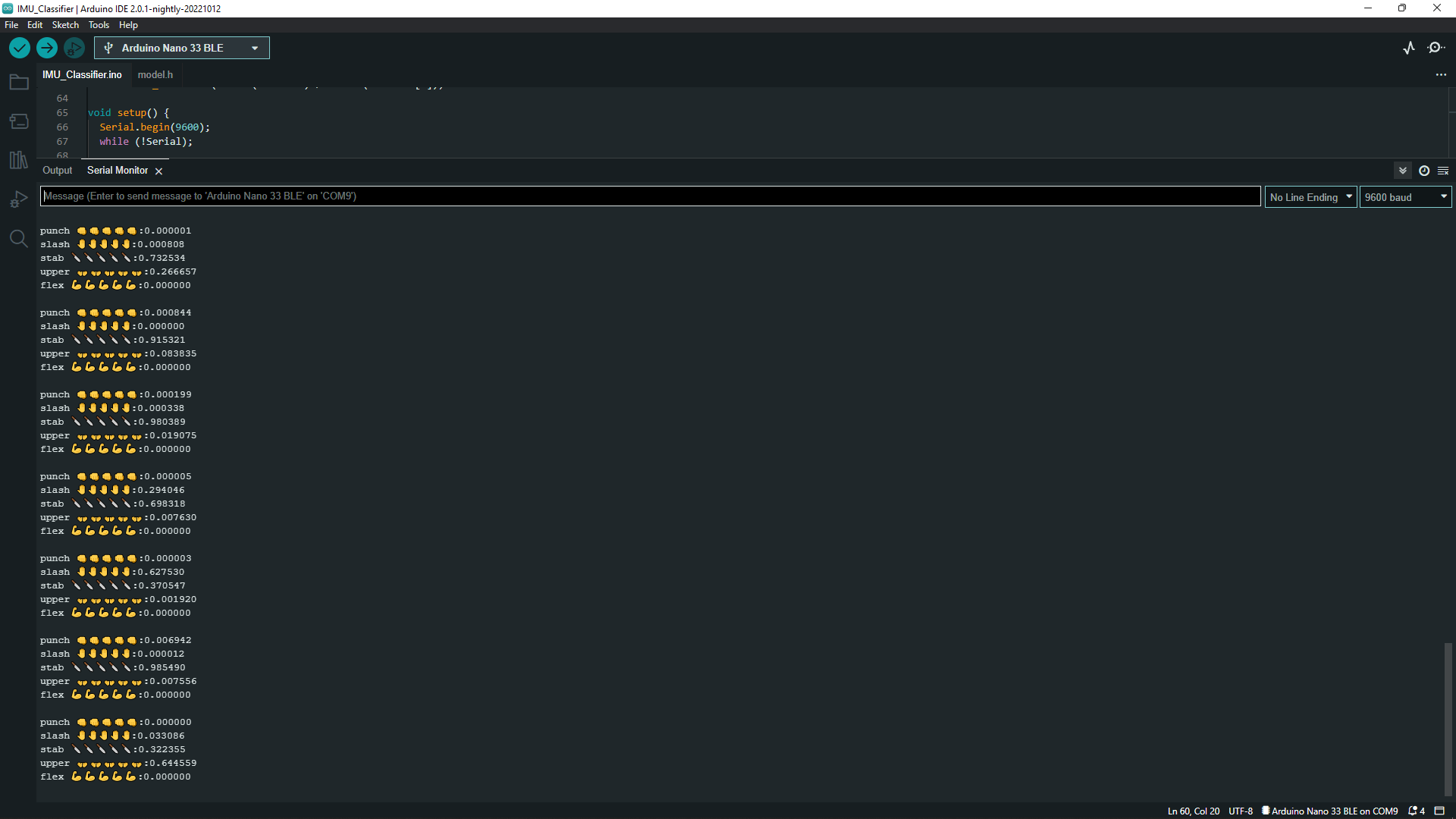
    }

  }

}

**Results**

When the code above is combined with the new model.h file which now contains 5 gestures including punch, flex, stab, slash and upper, the results are seen below:

****

**Project Setbacks**

After carefully and strategically observing the hand gestures with respect to the probabilities as detected by the Arduino Nano 33 bLE, there are some slight inaccuracies with the hand gestures especially the flex gesture.

**Possible Solutions**

Carefully retraining the flex and other inaccurate gesture to improve it’s recognizability. This should be done thoroughly to ensure maximum precision and accuracy avoiding any errors due to inaccurate or sharp hand movements, air resistance among others.

**Note**

As observed, several other hand gestures were recognized since we trained and included them in the data later on (model.h).

**Conclusion**

At the end of this project, we concluded that if such a project is further developed using tech gloves and other means of technology, one’s gestures can be accurately recognized and applied in areas like gyms and fitness as well as remote physiotherapy.

**Note:** Further development of this project is done in the next report (October 31st - DAY 9(Week 4) by connecting the hand gesture data in the serial monitor to other third party softwares like Excel to effectively gather and organize the data.

**References**

[1] <https://github.com/arduino/ArduinoTensorFlowLiteTutorials/>

[2] <https://docs.arduino.cc/tutorials/nano-33-ble-sense/get-started-with-machine-learning>

[3]<https://colab.research.google.com/github/arduino/ArduinoTensorFlowLiteTutorials/blob/master/GestureToEmoji/arduino_tinyml_workshop.ipynb#scrollTo=9J33uwpNtAku>