****

(All rights reserved)

**Interns**: Evans Acheampong and Josiah Lansah

**Course of study**: Computer Engineering

# Instructor: Douglas T. Ayitey

# Date: October 19th, 2022

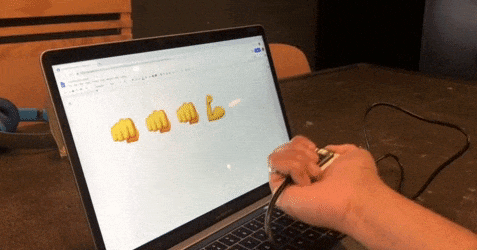
**BEST PROJECT REPORT**

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

**GESTURE RECOGNITION USING ARDUINO NANO 33 BLE SENSE**

**Introduction**

This project involves training your own custom gesture recognition model for Arduino using TensorFlow in Colab. This material is based on a practical workshop held by Sandeep Mistry and Don Coleman. Based on our research, we found that the following project is based on TensorFlow Lite for Microcontrollers which is currently experimental within the [TensorFlow repo](https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/experimental/micro). This is still a new and emerging field!



## Goals of this project

* Learn the fundamentals of TinyML implementation and training.
* Use the [Arduino\_LSM9DS1](https://www.arduino.cc/en/Reference/ArduinoLSM9DS1) and [Arduino\_TensorFlowLite](https://www.arduino.cc/reference/en/libraries/arduino_tensorflowlite/) libraries

**Project Setbacks**

Just like the previous machine learning project (Magic Wand), this gesture recognition project also had similar setbacks.

* The gesture recognition project requires the use of two major libraries: Arduino\_LSM9DS1 and Arduino\_TensorFlowLite.
* The first library had no known issues whatsoever. Unfortunately, the Arduino\_TensorFlowLite library is no more available in any version of Arduino and hence we resulted to the use of Arduino Nano 33 BLE Sense board since Arduino Nano RP2040 is relatively new in the system, it still has some unresolved issues one of which is machine learning applications especially involving TensorFlow lite.
* Hence Arduino Nano 33 BLE Sense board was employed in this project instead.

## Hardware & Software Needed

* 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).

The Arduino Nano 33 BLE Sense has a variety of onboard sensors meaning potential for some cool TinyML applications:

* Voice – digital microphone
* Motion – 9-axis IMU (accelerometer, gyroscope, magnetometer)
* Environmental – temperature, humidity and pressure
* Light – brightness, color and object proximity

Unlike classic Arduino Uno, the board combines a microcontroller with onboard sensors which means you can address many use cases without additional hardware or wiring. The board is also small enough to be used in end applications like wearables. As the name suggests it has Bluetooth® Low Energy connectivity so you can send data (or inference results) to a laptop, mobile app or other Bluetooth® Low Energy boards and peripherals.

**Things we noted**

* **Sensors on a USB stick** – Connecting the BLE Sense board over USB is an easy way to capture data and add multiple sensors to single board computers without the need for additional wiring or hardware
* Machine learning can make microcontrollers accessible to developers who don’t have a background in embedded development (as in our case).
* On the machine learning side, there are techniques you can use to fit neural network models into memory constrained devices like microcontrollers. One of the key steps is the [quantization of the weights](https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/) from floating point to 8-bit integers. This also has the effect of making inference quicker to calculate and more applicable to lower clock-rate devices.
* TinyML is an emerging field and there is still work to do – but what’s exciting is there’s a vast unexplored application space out there. Billions of microcontrollers combined with all sorts of sensors in all sorts of places which can lead to some seriously creative and valuable TinyML applications in the future.

## **Training a TensorFlow Lite Micro Model For Arduino**

We used ML to enable the Arduino board to recognize gestures. We captured motion data from the Arduino Nano 33 BLE Sense board, imported it into TensorFlow to train a model, and deployed the resulting classifier onto the board.

Since we used the offline IDE, we needed to install the driver manually. This can be done by navigating to **Tools > Board > Board Manager...**, search for **Arduino Mbed OS Nano Boards**, and install it.

Also, the other drivers can be downloading by navigating to **Tools > Manage libraries...**, search for **Arduino\_TensorFlowLite** and **Arduino:LSM9DS1**, and install them both.

As said earlier the **Arduino\_TensorFlowLite library is currently not available and hence third-party versions were used. The was provided in the previous report as well as in the reference below**

## **Streaming Sensor Data From the Arduino Board**

First, we needed to capture some training data. We captured sensor data logs from the Arduino board over the same USB cable you use to program the board with the laptop or PC.

Arduino boards run small applications (also called sketches) which are compiled from .ino format Arduino source code, and programmed onto the board using the Arduino IDE or Arduino Create.

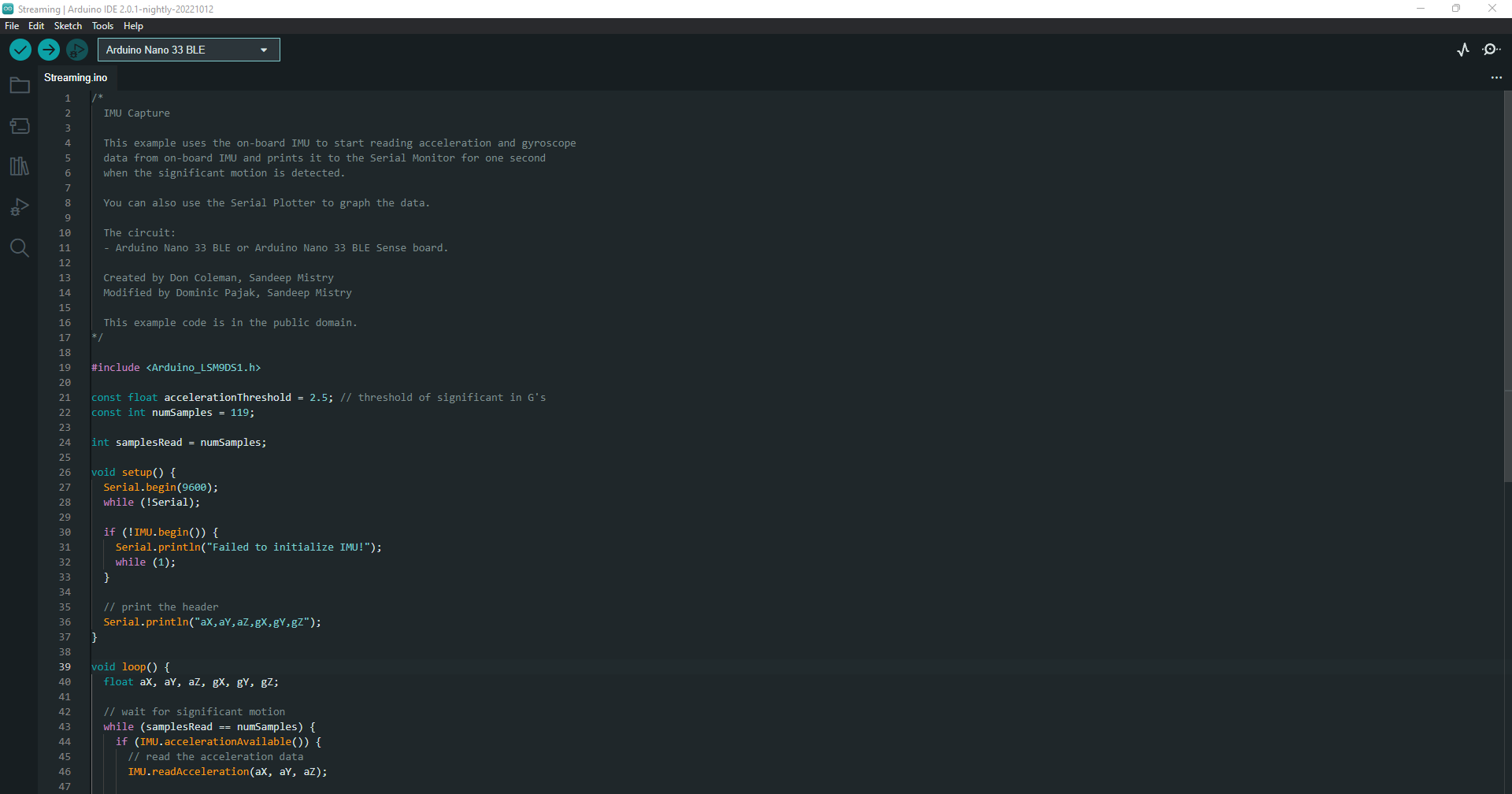
With the sketch we created we did the following:

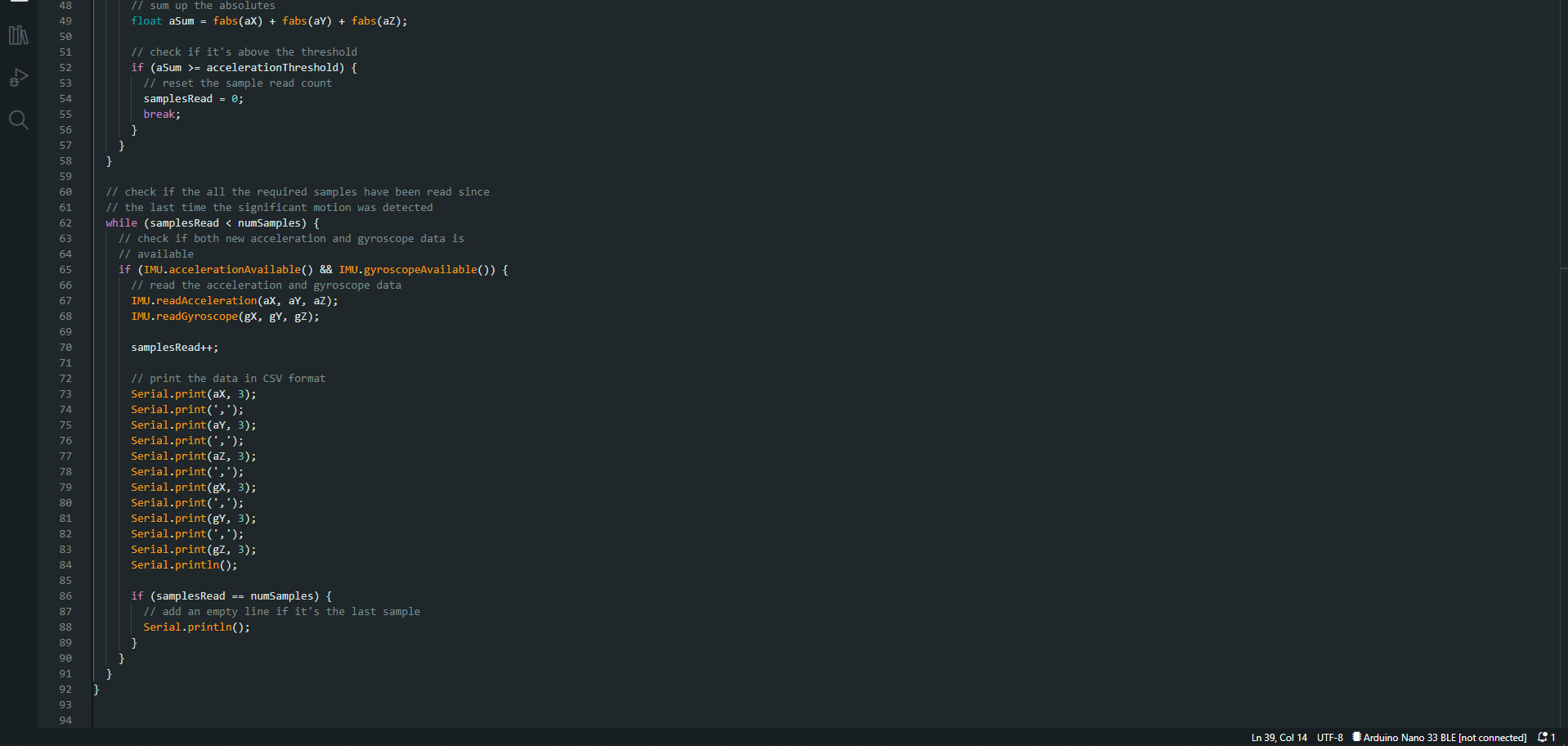
* Monitor the board’s accelerometer and gyroscope
* Trigger a sample window on detecting significant linear acceleration of the board
* Sample for one second at 119Hz, outputting CSV format data over USB
* Loop back and monitor for the next gesture

**Things we noted**

**The sensors we chose to read from the board, the sample rate, the trigger threshold, and whether we stream data output as CSV, JSON, binary or some other format are all customizable in the sketch running on the Arduino. There is also scope to perform signal preprocessing and filtering on the device before the data is output to the log**

The complete sketch can be found below:





## **Visualizing Live Sensor Data Log From the Arduino Board**

With that done we can now visualize the data coming off the board. We’re not capturing data yet this is just to give you a feel for how the sensor data capture is triggered and how long a sample window is. This will help when it comes to collecting training samples.

* In the Arduino IDE, open the Serial Plotter **Tools > Serial Plotter**
* Pick up the board and practice your punch ,flex, slash, stab and upper gestures.
* You’ll see it only sample for a one second window, then wait for the next gesture
* You should see a live graph of the sensor data capture (see Image below)

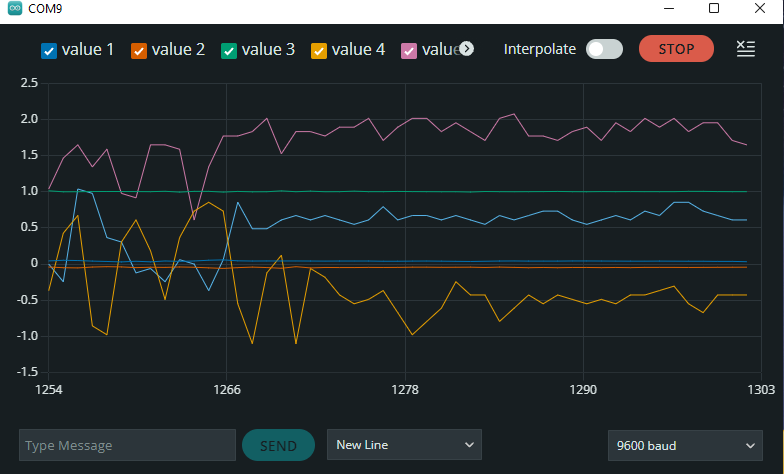
**Things we noted**

We encountered an error that the board is not available.

**Solution**

Reselect the port: **Tools > Port > portname (Arduino Nano 33 BLE)**

**Result**



**Things we noted**

**When we finished we has to close the Serial Plotter window – this is important as the next step won’t work otherwise.**

**SUMMARY**

* Project Setbacks

## Training a TensorFlow Lite Micro Model For Arduino

* Streaming Sensor Data From the Arduino Board

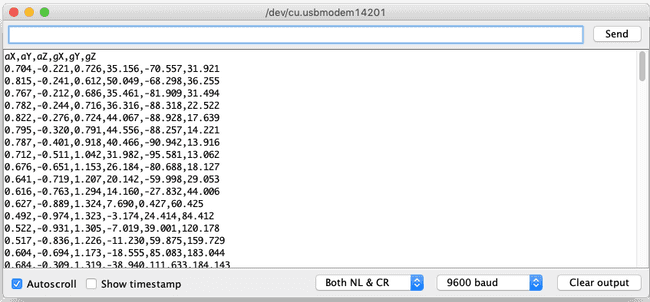
## Visualizing Live Sensor Data Log From the Arduino Board

## **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:

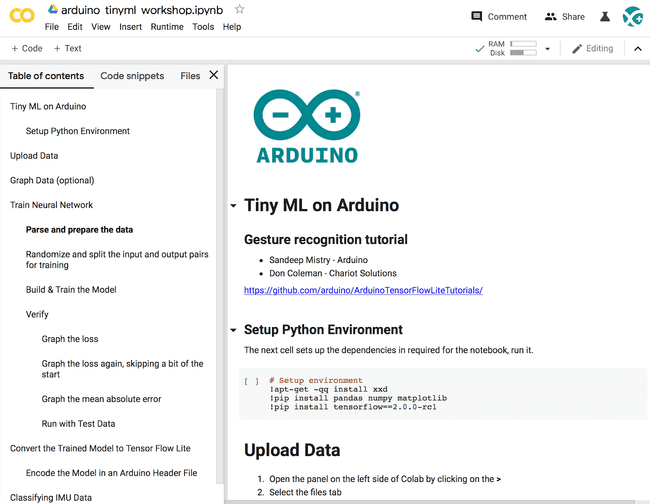
* Reset the board by pressing the small white button on the top
* Pick up the board in one hand (picking it up later will trigger sampling)
* In the Arduino IDE, open the Serial Monitor Tools > Serial Monitor
* If you get an error that the board is not available, reselect the port:
* Tools > Port > portname (Arduino Nano 33 BLE) [**This issue was encountered previously hence was easily detected and fixed**]!
* Make a punch gesture with the board in your hand (Be careful whilst doing this!)
* Make the outward punch 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 a flex gesture in a file called flex.csv
* Make the inward flex fast enough to trigger capture returning slowly each time

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

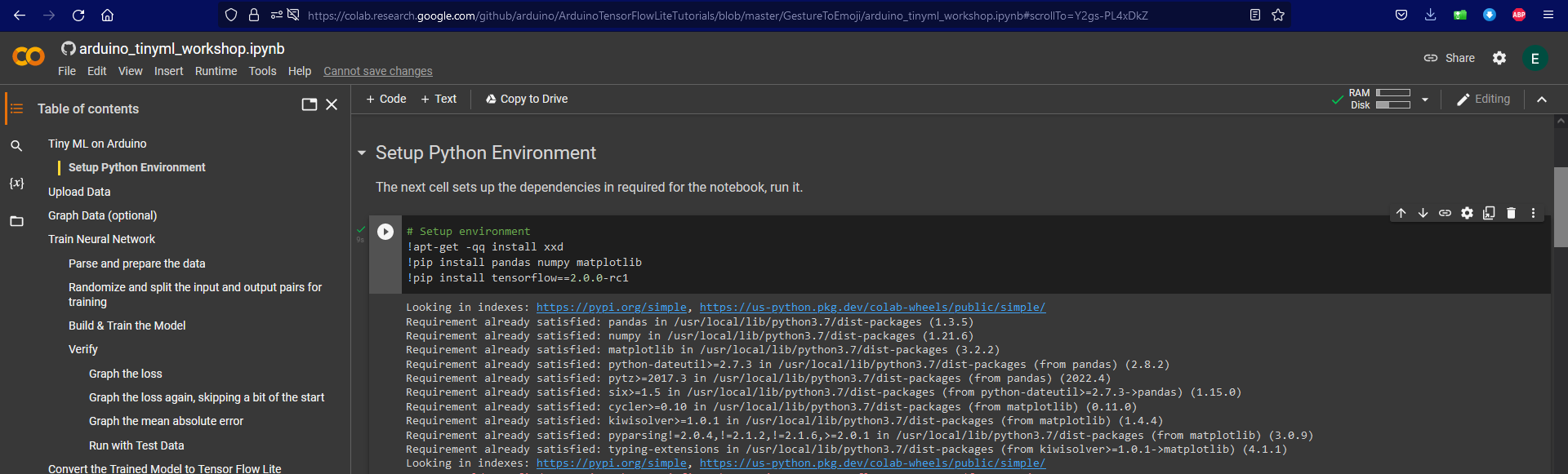


## Training in TensorFlow

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.



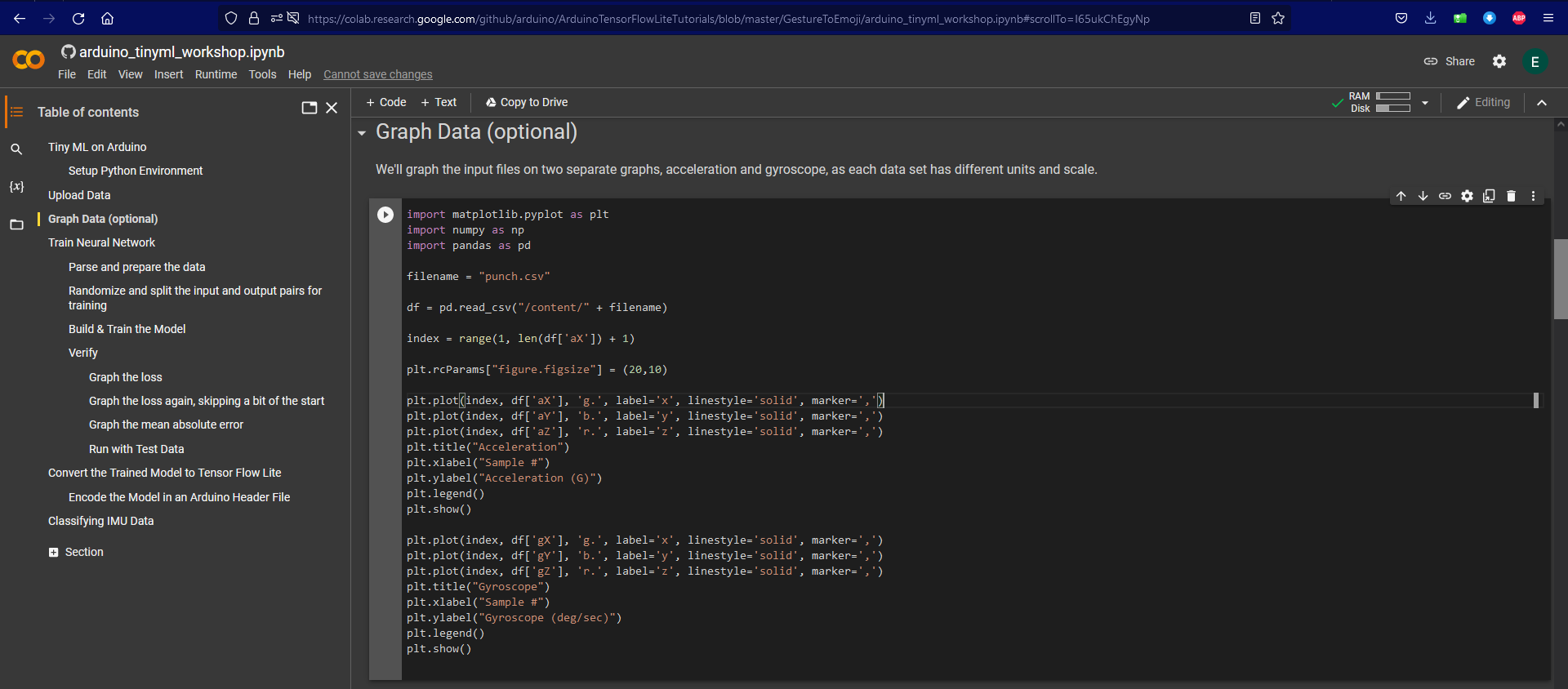
**Step 1: Set up the Python Environment**

****

## **Step 2: Upload the Data**

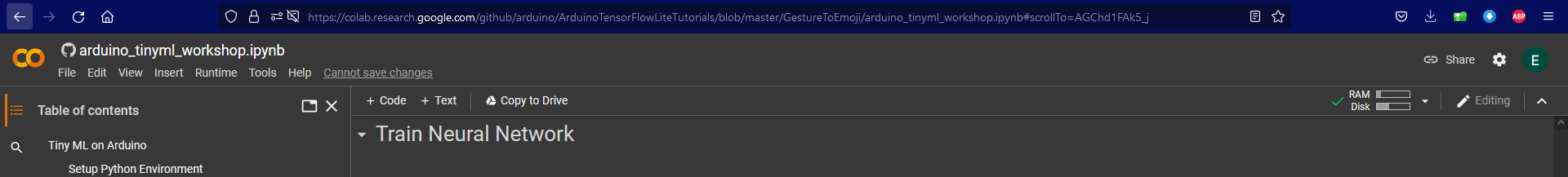
## 

## **Step 3: Graph Data (Optional)**



## 

## **Step 4: Build and Train the neural Network**



## **Step 5: Parse and prepare the data**

The next cell parses the csv files and transforms them to a format that will be used to train the fully connected neural network.

Update the GESTURES list with the gesture data you've collected in .csv format.

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import tensorflow as tf

print(f"TensorFlow version = {tf.\_\_version\_\_}\n")

# Set a fixed random seed value, for reproducibility, this will allow us to get

# the same random numbers each time the notebook is run

SEED = 1337

np.random.seed(SEED)

tf.random.set\_seed(SEED)

# the list of gestures that data is available for

GESTURES = [

    "punch",

    "flex",

]

SAMPLES\_PER\_GESTURE = 119

NUM\_GESTURES = len(GESTURES)

# create a one-hot encoded matrix that is used in the output

ONE\_HOT\_ENCODED\_GESTURES = np.eye(NUM\_GESTURES)

inputs = []

outputs = []

# read each csv file and push an input and output

for gesture\_index in range(NUM\_GESTURES):

  gesture = GESTURES[gesture\_index]

  print(f"Processing index {gesture\_index} for gesture '{gesture}'.")

  output = ONE\_HOT\_ENCODED\_GESTURES[gesture\_index]

  df = pd.read\_csv("/content/" + gesture + ".csv")

  # calculate the number of gesture recordings in the file

  num\_recordings = int(df.shape[0] / SAMPLES\_PER\_GESTURE)

  print(f"\tThere are {num\_recordings} recordings of the {gesture} gesture.")

  for i in range(num\_recordings):

    tensor = []

    for j in range(SAMPLES\_PER\_GESTURE):

      index = i \* SAMPLES\_PER\_GESTURE + j

      # normalize the input data, between 0 to 1:

      # - acceleration is between: -4 to +4

      # - gyroscope is between: -2000 to +2000

      tensor += [

          (df['aX'][index] + 4) / 8,

          (df['aY'][index] + 4) / 8,

          (df['aZ'][index] + 4) / 8,

          (df['gX'][index] + 2000) / 4000,

          (df['gY'][index] + 2000) / 4000,

          (df['gZ'][index] + 2000) / 4000

      ]

    inputs.append(tensor)

    outputs.append(output)

# convert the list to numpy array

inputs = np.array(inputs)

outputs = np.array(outputs)

print("Data set parsing and preparation complete.")

## **Step 6: Randomize and split the input and output pairs for training**

Randomly split input and output pairs into sets of data: 60% for training, 20% for validation, and 20% for testing.

* the training set is used to train the model
* the validation set is used to measure how well the model is performing during training
* the testing set is used to test the model after training

# Randomize the order of the inputs, so they can be evenly distributed for training, testing, and validation

# https://stackoverflow.com/a/37710486/2020087

num\_inputs = len(inputs)

randomize = np.arange(num\_inputs)

np.random.shuffle(randomize)

# Swap the consecutive indexes (0, 1, 2, etc) with the randomized indexes

inputs = inputs[randomize]

outputs = outputs[randomize]

# Split the recordings (group of samples) into three sets: training, testing and validation

TRAIN\_SPLIT = int(0.6 \* num\_inputs)

TEST\_SPLIT = int(0.2 \* num\_inputs + TRAIN\_SPLIT)

inputs\_train, inputs\_test, inputs\_validate = np.split(inputs, [TRAIN\_SPLIT, TEST\_SPLIT])

outputs\_train, outputs\_test, outputs\_validate = np.split(outputs, [TRAIN\_SPLIT, TEST\_SPLIT])

print("Data set randomization and splitting complete.")

## **Step 7: Build & Train the Model**

Build and train a [TensorFlow](https://www.tensorflow.org) model using the high-level [Keras](https://www.tensorflow.org/guide/keras) API.

# build the model and train it

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(50, activation='relu')) # relu is used for performance

model.add(tf.keras.layers.Dense(15, activation='relu'))

model.add(tf.keras.layers.Dense(NUM\_GESTURES, activation='softmax')) # softmax is used, because we only expect one gesture to occur per input

model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

history = model.fit(inputs\_train, outputs\_train, epochs=600, batch\_size=1, validation\_data=(inputs\_validate, outputs\_validate))

## **Step 8: Verify**

Graph the model’s performance vs validation.

# Step 9: Convert the Trained Model to Tensor Flow Lite

The next cell converts the model to TFlite format. The size in bytes of the model is also printed out.

# Convert the model to the TensorFlow Lite format without quantization

converter = tf.lite.TFLiteConverter.from\_keras\_model(model)

tflite\_model = converter.convert()

# Save the model to disk

open("gesture\_model.tflite", "wb").write(tflite\_model)

import os

basic\_model\_size = os.path.getsize("gesture\_model.tflite")

print("Model is %d bytes" % basic\_model\_size)

## **Step 10: Encode the Model in an Arduino Header File**

The next cell creates a constant byte array that contains the TFlite model. Import it as a tab with the sketch below.

!echo "const unsigned char model[] = {" > /content/model.h

!cat gesture\_model.tflite | xxd -i      >> /content/model.h

!echo "};"                              >> /content/model.h

import os

model\_h\_size = os.path.getsize("model.h")

print(f"Header file, model.h, is {model\_h\_size:,} bytes.")

print("\nOpen the side panel (refresh if needed). Double click model.h to download the file.")

# Step 11: Classifying IMU Data

Now it's time to switch back to the tutorial instructions and run our new model on the Arduino Nano 33 BLE Sense to classify the accelerometer and gyroscope data.

## **Step 12: Classifying IMU Data**

Next, we will use model.h file we just trained and downloaded from Colab in the previous section in our Arduino IDE project:

We will be starting a new sketch, you will find the complete code below:

/\*

  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/all\_ops\_resolver.h>

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

#include <tensorflow/lite/micro/micro\_interpreter.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",

  "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();

      }

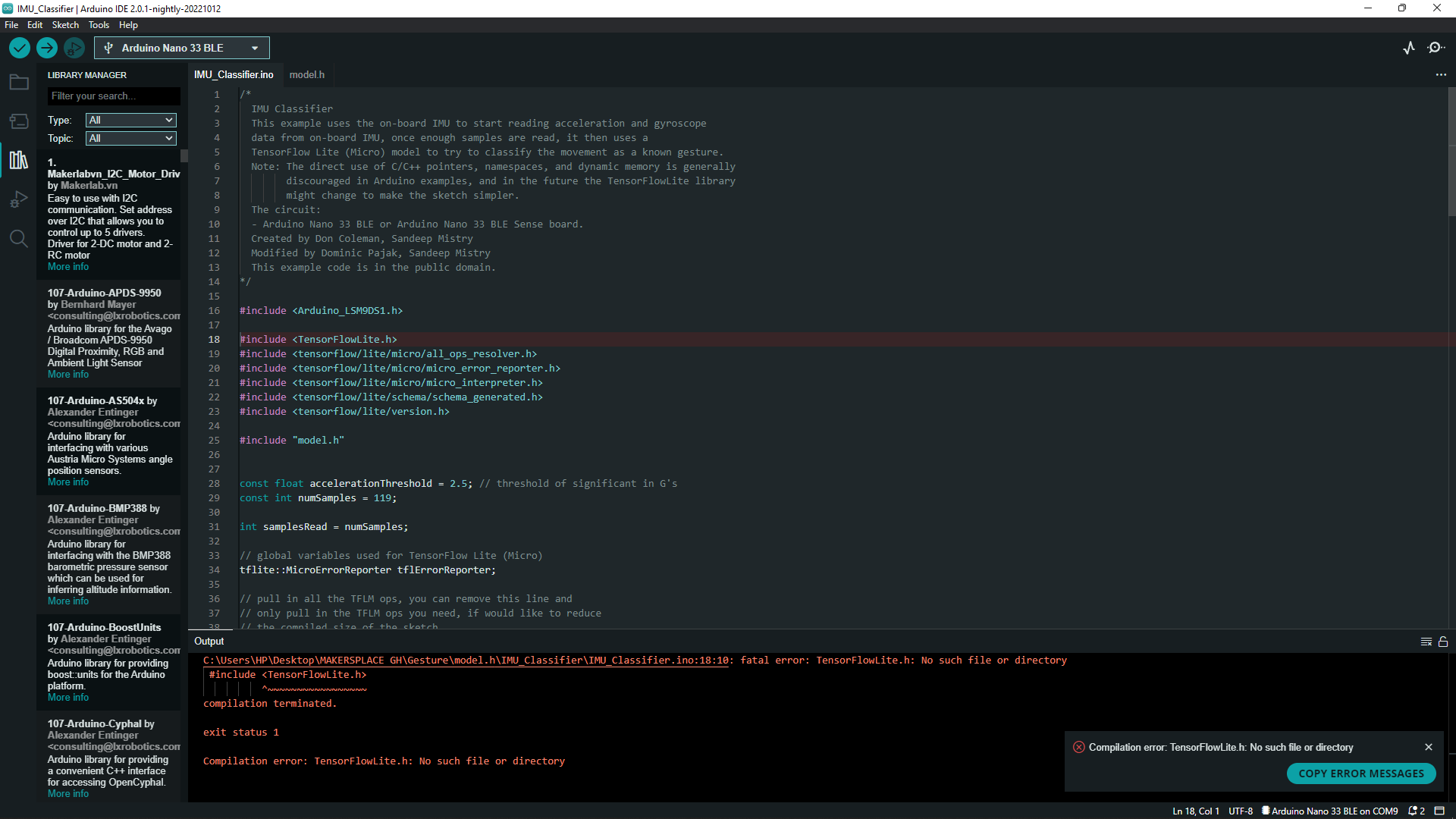
    }

  }

}

* Create a new tab in the IDE. When asked name it model.h
* Open the model.h tab and paste in the version you downloaded from Colab
* Upload the sketch: Sketch > Upload
* Open the Serial Monitor: Tools > Serial Monitor
* Perform some gestures
* The confidence of each gesture will be printed to the Serial Monitor (0 = low confidence, 1 = high confidence)
* For added fun the [Emoji\_Button.ino](https://github.com/arduino/ArduinoTensorFlowLiteTutorials/blob/master/GestureToEmoji/ArduinoSketches/Emoji_Button/Emoji_Button.ino) example shows how to create a USB keyboard that prints an emoji character in Linux and macOS. Combining the [Emoji\_Button.ino](https://github.com/arduino/ArduinoTensorFlowLiteTutorials/blob/master/GestureToEmoji/ArduinoSketches/Emoji_Button/Emoji_Button.ino) example with the [IMU\_Classifier.ino](https://github.com/arduino/ArduinoTensorFlowLiteTutorials/blob/master/GestureToEmoji/ArduinoSketches/IMU_Classifier/IMU_Classifier.ino) sketch to create a gesture controlled emoji keyboard involves another separate task which would be duly done.

**Errors**

****

* The error was due to the absence of the in-built Arduino\_TensorFlowLite library. Using any of the alternative libraries such as **Adafruit\_TensorFlow\_Lite, tensorflow\_lite, TensorFlowLite\_ESP32, EloquentTinyML.** gave the Compilation error: TensorFlowLite.h: No such file or directory.
* Also using **Arduino\_TensorFlowLite-2.4.0-ALPHA-precompiled library** worked for other machine learning projects such as the magic wand but failed for the gesture recognition due to **undefined reference to `DebugLog’ error**

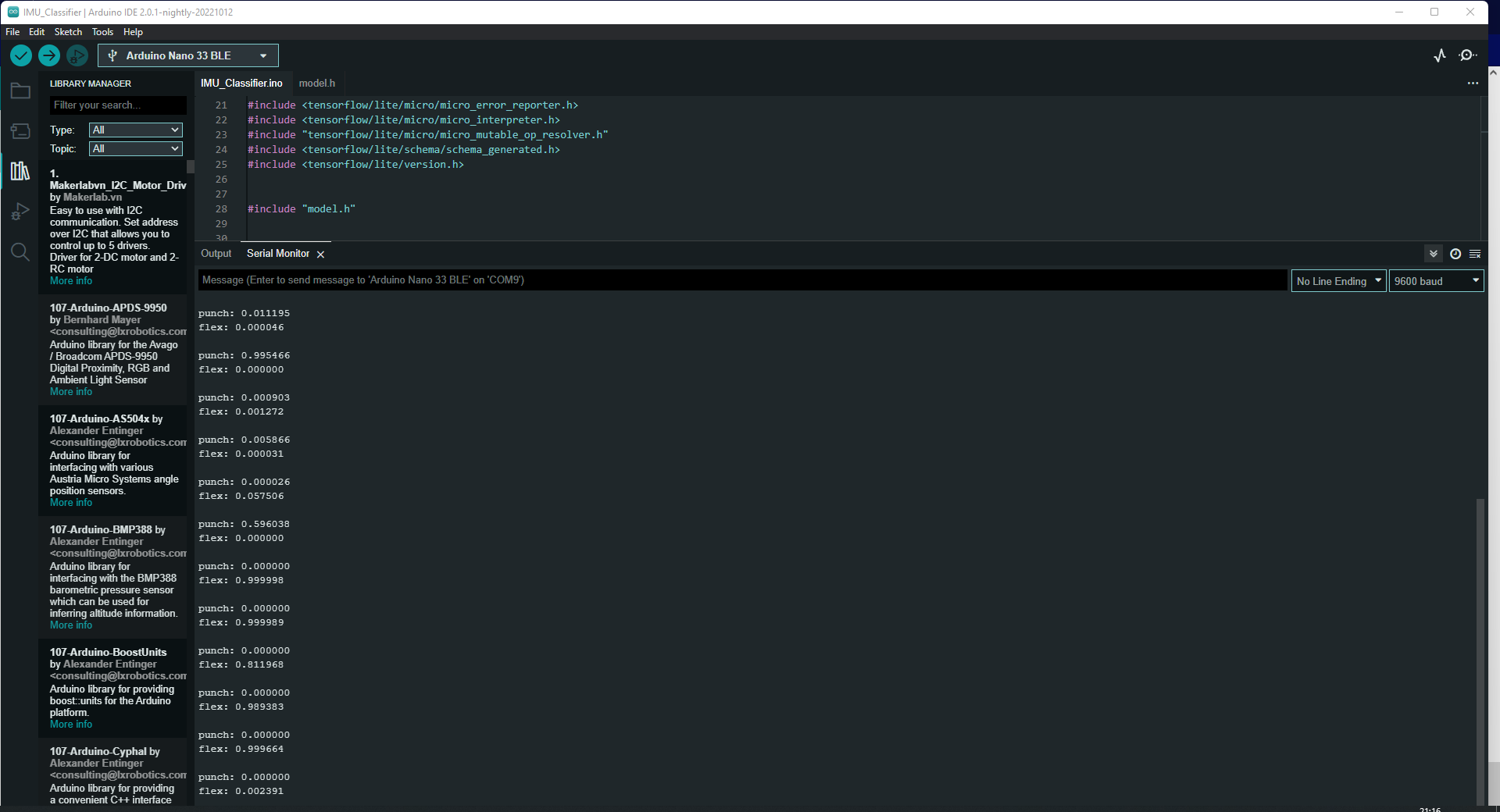
**Possible Solutions**

Through vigorous searches and research, we finally found a library that gives no errors whatsoever and fully supports the Arduino Nano 33 BLE Sense**: Ameba\_TensorFlowLite**

**Given in the link below:**

[**https://github.com/ambiot/ambd\_arduino/blob/master/Arduino\_zip\_libraries/Ameba\_TensorFlowLite.zip**](https://github.com/ambiot/ambd_arduino/blob/master/Arduino_zip_libraries/Ameba_TensorFlowLite.zip)

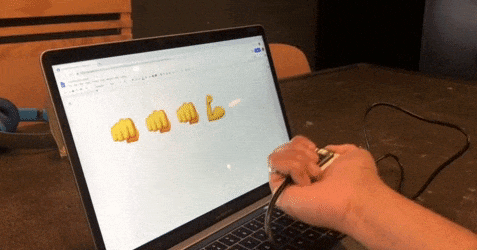
**Results**

****

## **Conclusion**

* It’s an exciting time with a lot to learn and explore in TinyML. This project has given us some idea of the potential and a starting point to start applying it in our own projects.
* Efforts for the next project I going to involve the use of Emojis for the various punch and flex gestures.

This stage involves the addition of the emoji button to the gesture recognition project which was finalized in the former report as shown in the diagram below:



**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).
* Ubuntu
* Virtual Machine Ware

**Goals and Objectives**

* For added fun the [Emoji\_Button.ino](https://github.com/arduino/ArduinoTensorFlowLiteTutorials/blob/master/GestureToEmoji/ArduinoSketches/Emoji_Button/Emoji_Button.ino) example shows how to create a USB keyboard that prints an emoji character in Linux and macOS. We tried combining the [Emoji\_Button.ino](https://github.com/arduino/ArduinoTensorFlowLiteTutorials/blob/master/GestureToEmoji/ArduinoSketches/Emoji_Button/Emoji_Button.ino) example with the [IMU\_Classifier.ino](https://github.com/arduino/ArduinoTensorFlowLiteTutorials/blob/master/GestureToEmoji/ArduinoSketches/IMU_Classifier/IMU_Classifier.ino) sketch to create a gesture controlled emoji keyboard.
* The code used is given as shown below:
* /\*
* Emoji Button
* This example sends an emoji character over USB HID when the button is pressed.
* Note: Only macOS and Linux as supported at this time, and the use of
* #define is generally discouraged in Arduino examples
* The circuit:
* - Arduino Nano 33 BLE or Arduino Nano 33 BLE Sense board.
* - Button connected to pin 3 and GND.
* Created by Don Coleman, Sandeep Mistry
* This example code is in the public domain.
* \*/
* #include <PluggableUSBHID.h>
* #include <USBKeyboard.h>
* // Select an OS:
* //#define MACOS // You'll need to enable and select the unicode keyboard: System Preferences -> Input Sources -> + -> Others -> Unicode Hex Input
* //#define LINUX
* #if !defined(MACOS) && !defined(LINUX)
* #error "Please select an OS!"
* #endif
* // use table: https://apps.timwhitlock.info/emoji/tables/unicode
* const int bicep = 0x1f4aa;
* const int punch = 0x1f44a;
* const int buttonPin = 3;
* USBKeyboard keyboard;
* int previousButtonState = HIGH;
* void setup() {
* pinMode(buttonPin, INPUT\_PULLUP);
* }
* void loop() {
* int buttonState = digitalRead(buttonPin);
* if (buttonState != previousButtonState) {
* if (buttonState == LOW) {
* // pressed
* sentUtf8(bicep);
* } else {
* // released
* }
* previousButtonState = buttonState;
* }
* }
* void sentUtf8(unsigned long c) {
* String s;
* #if defined(MACOS)
* // https://apple.stackexchange.com/questions/183045/how-can-i-type-unicode-characters-without-using-the-mouse
* s = String(utf8ToUtf16(c), HEX);
* for (int i = 0; i < s.length(); i++) {
* keyboard.key\_code(s[i], KEY\_ALT);
* }
* #elif defined(LINUX)
* s = String(c, HEX);
* keyboard.key\_code('u', KEY\_CTRL | KEY\_SHIFT);
* for (int i = 0; i < s.length(); i++) {
* keyboard.key\_code(s[i]);
* }
* #endif
* keyboard.key\_code(' ');
* }
* // based on https://stackoverflow.com/a/6240819/2020087
* unsigned long utf8ToUtf16(unsigned long in) {
* unsigned long result;
* in -= 0x10000;
* result |= (in & 0x3ff);
* result |= (in << 6) & 0x03ff0000;
* result |= 0xd800dc00;
* return result;
* }

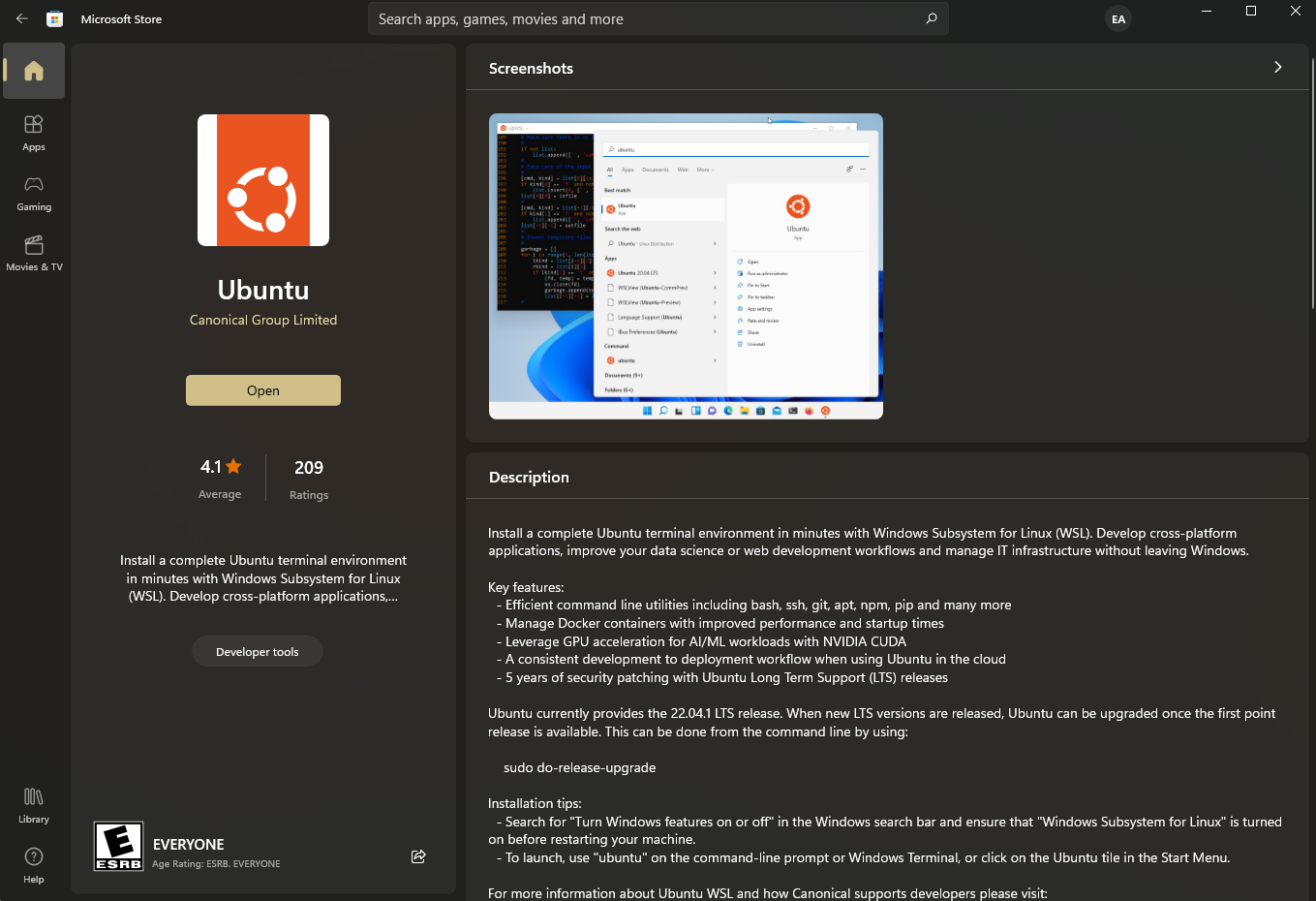
**Project Setbacks**

* In order to add the emoji button to the project using the code above, a Linux or macOS is required. This therefore counted as a major setback especially due to the fact that we were mostly limited to Windows Operating systems.

**Possible Solutions**

* Using a Linux based operating system.

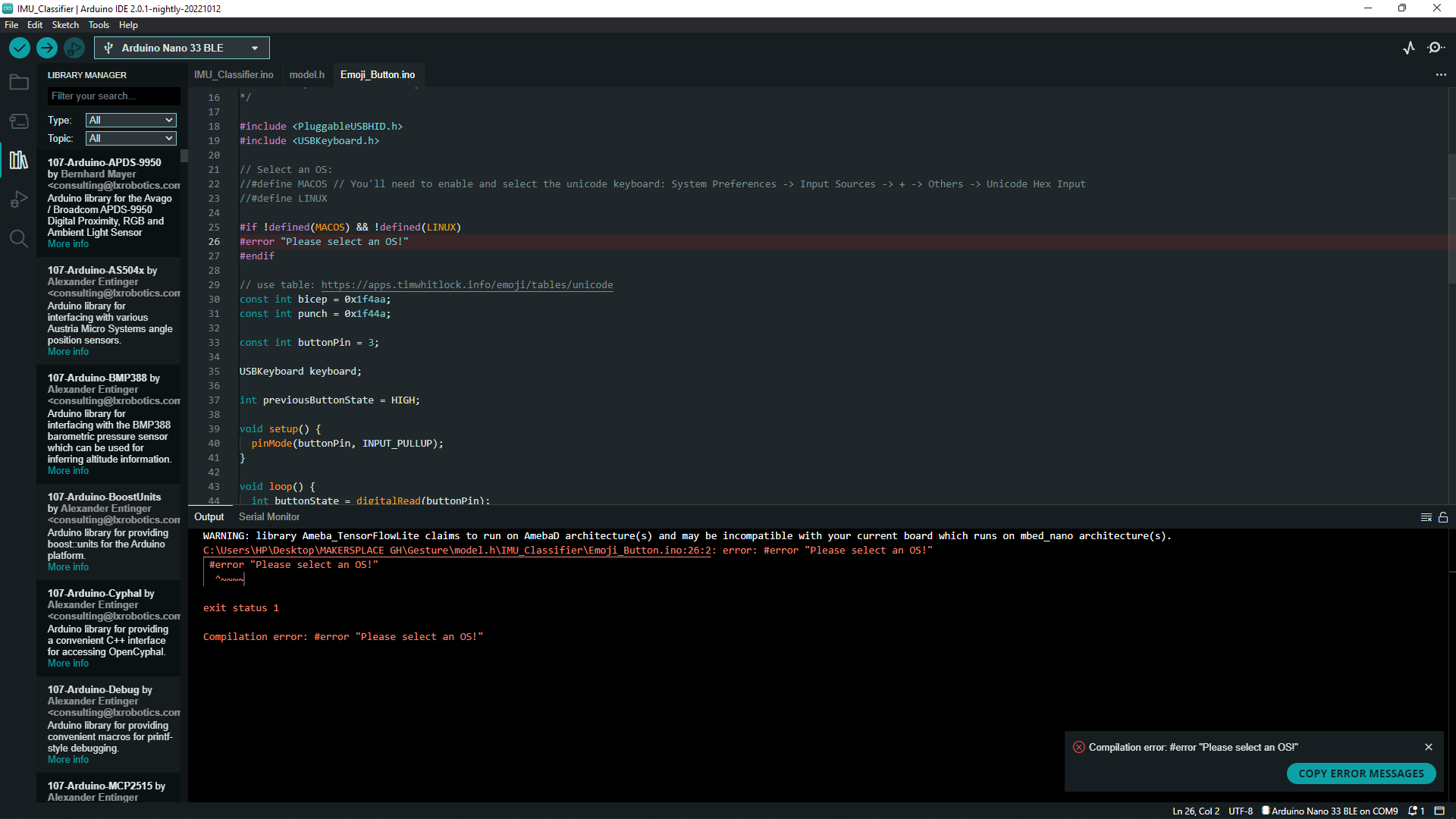
In order to use a Linux OS, Ubuntu was obtained from the Microsoft Store.



**Procedure**

* Import the [Emoji\_Button.ino](https://github.com/arduino/ArduinoTensorFlowLiteTutorials/blob/master/GestureToEmoji/ArduinoSketches/Emoji_Button/Emoji_Button.ino) into the code alongside the IMU Classifier and model.h code in the Arduino IDE.

**Project Setbacks and Errors**

****

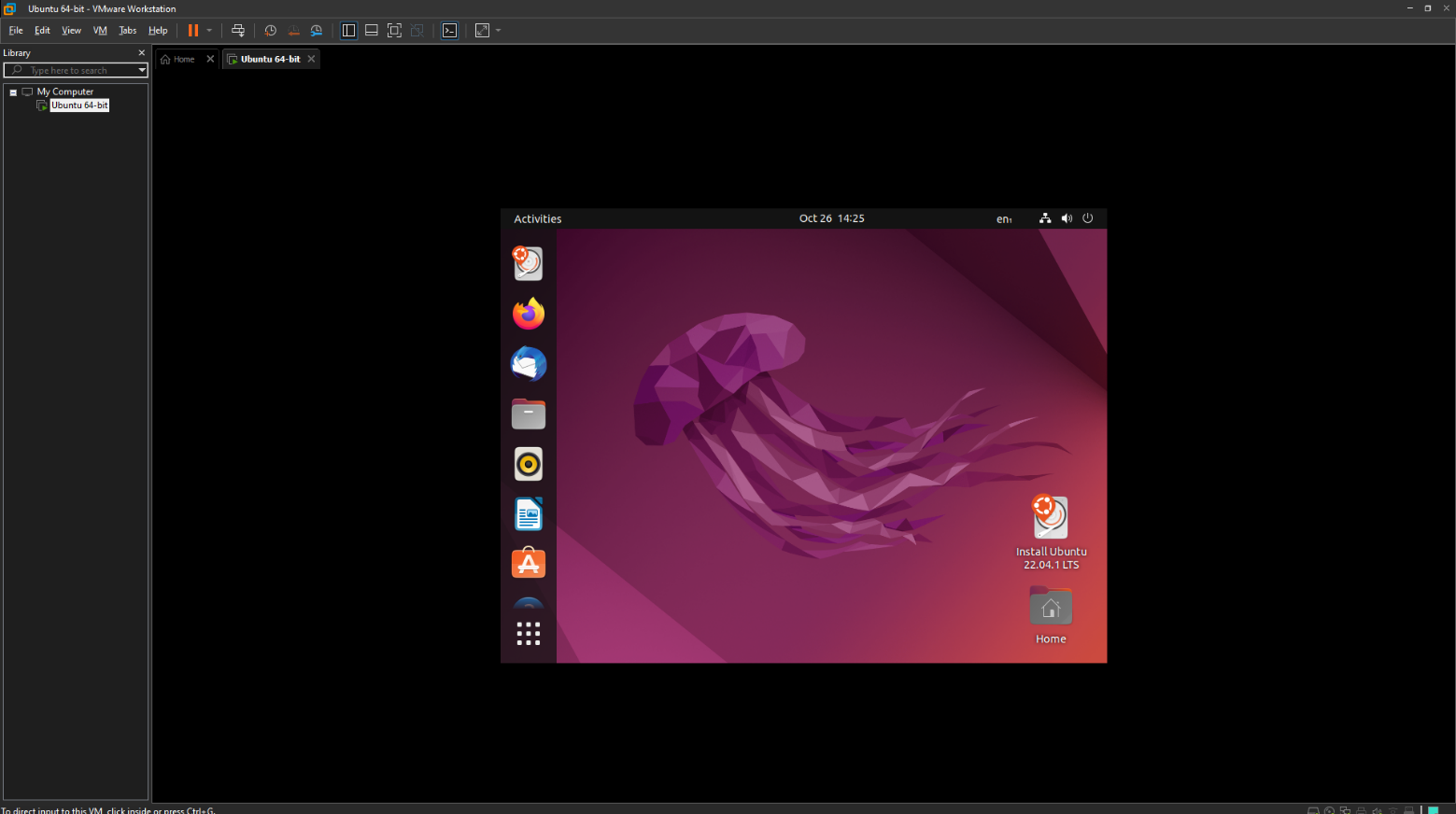
* After the installation of Ubuntu 20.04, an error was encountered as shown above.

Error: "Please select an OS!"

* Possible Solutions

Using VM Ware

## After installing and setting up the Virtual Machine Ware, we managed to create and **Install Ubuntu Linux On VMWare Workstation as shown below:**



**Project Setbacks**

After hours of working with the ubuntu interface on VM Ware, we encountered several setbacks and errors:

* Since the Linux OS is running on a virtual machine, it is very slow, laggy and thus requires lots of time and effort.
* Downloading and running the Arduino ide on virtual machine takes a considerable amount of time and effort due to low memory allocation.
* Compilation several lines of code using the Arduino ide on a virtual machine is a very slow and time-consuming process which may result in errors.

**Possible Solutions**

After careful consideration of the factors above, we decided to make use of another platform –

Mac OS. Due to its high speed and readability, The Macintosh would be an appropriate alternative to Ubuntu which was initially run on a virtual platform.

NOTE: It must be well noted that this does not entail the entire the added feature of Emoji recognition project but about the first phase of it. The finalization of this project is detailed in the document of October 26th - DAY 7(Week 3).

## **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>

[4] <https://www.ardu-badge.com/Arduino_TensorFlowLite/zip>

[5]<https://github.com/ambiot/ambd_arduino/blob/master/Arduino_zip_libraries/Ameba_TensorFlowLite.zip>

[6]<https://github.com/arduino/ArduinoTensorFlowLiteTutorials/blob/master/GestureToEmoji/ArduinoSketches/Emoji_Button/Emoji_Button.ino>

[7] <https://blog.tensorflow.org/2019/11/how-to-get-started-with-machine.html>

[8] <https://thesecmaster.com/install-ubuntu-linux-on-vmware-workstation/>