**Edge Computing Laboratory**

**Lab Assignment 8**

**Name:** Gaurav Gadekar

**Class:** TY AIEC Batch C

**Enrollment No:** MITU23BTCSD120

**Roll No:** D2233120

### **Title The "magic wand" project that can recognize gestures using an accelerometer and an ML classification model on Edge Devices**

**Objective:** Build a project to detect the accelerometer values and convert them into gestures **Tasks:**

* Generate the dataset for Accelerometer Motion (Up-Down, Left-Right)
* Configure BLE Sense / Mobile for Edge Impulse
* Building and Training a Model
* Deploy on Nano BLE Sense / Mobile Phone

### Introduction

Edge Impulse is a development platform for machine learning on edge devices, targeted at developers who want to create intelligent device solutions. The " Accelerometer Motion "sensor reading equivalent in Edge Impulse would typically involve creating a simple machine learning model that can run on an edge device, like classifying sensor data or recognizing a basic pattern.

Materials Required

• Nano BLE Sense Board

### Theory

GPIO (General Purpose Input/Output) pins on the Raspberry Pi are used for interfacing with other electronic components. BCM numbering refers to the pin numbers in the Broadcom SOC channel, which is a more consistent way to refer to the GPIO pins across different versions of the

Here’s a high-level overview of steps you'd follow to create a "Hello World" project on Edge Impulse:

**Steps to Configure the Edge Impulse:**

Create an Account and New Project:

* + Sign up for an Edge Impulse account.
  + Create a new project from the dashboard.

Connect a Device:

* + You can use a supported development board or your smartphone as a sensor device.
  + Follow the instructions to connect your device to your Edge Impulse project.

Collect Data:

Use the Edge Impulse mobile app or the Web interface to collect data from the onboard sensors.

* + For a "Hello World" project, you could collect accelerometer data, for instance.

Create an Impulse:

* + Go to the 'Create impulse' page.
  + Add a processing block (e.g., time-series data) and a learning block (e.g., classification).
  + Save the impulse, which defines the machine learning pipeline.

Design a Neural Network:

* + Navigate to the 'NN Classifier' under the 'Learning blocks'.
  + Design a simple neural network. Edge Impulse provides a default architecture that works well for most basic tasks.

Train the Model:

* + Click on the 'Start training' button to train your machine learning model with the collected data.

Test the Model:

* + Once the model is trained, you can test its performance with new data in the 'Model Testing' tab.

Deploy the Model:

* + Go to the 'Deployment' tab.
  + Select the deployment method that suits your edge device (e.g., Arduino library, WebAssembly, container, etc.).
  + Follow the instructions to deploy the model to your device.

Run Inference:

* + With the model deployed, run inference on the edge device to see it classifying data in real-time.

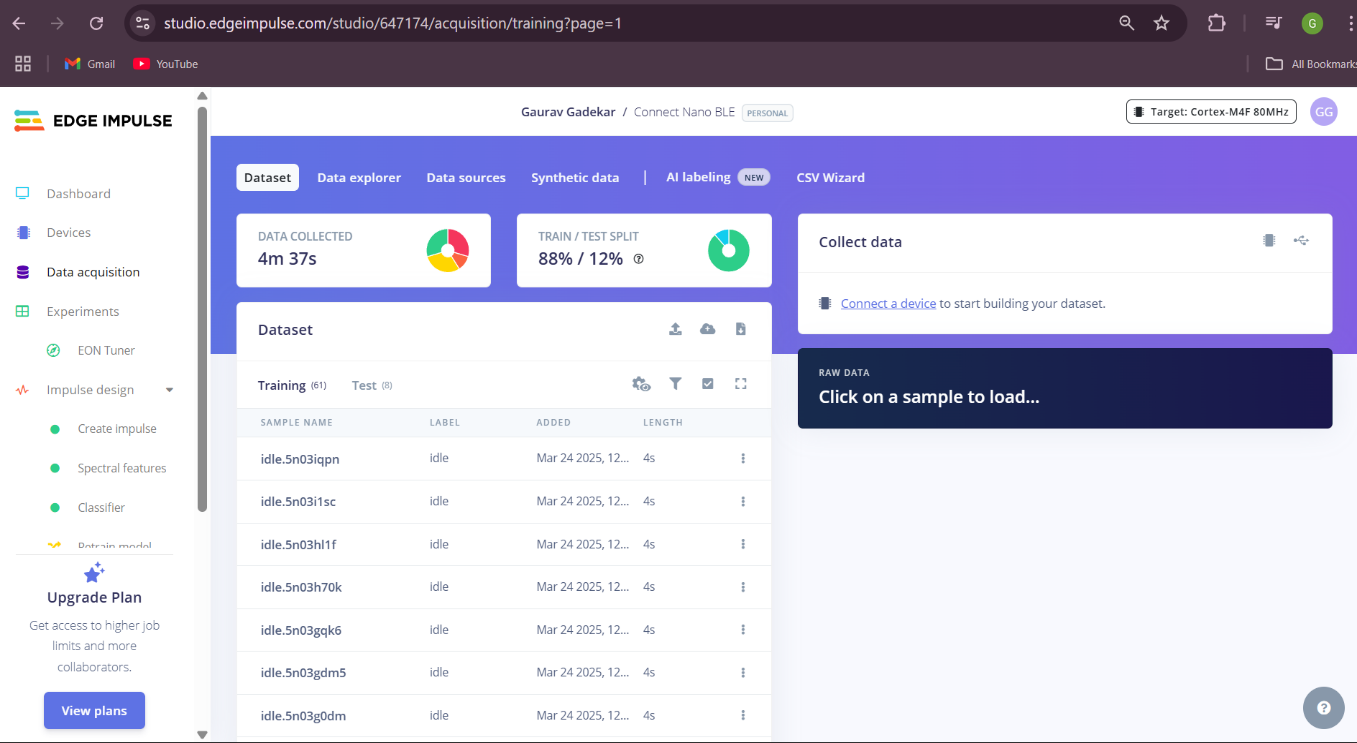
Monitor:

* + You can monitor the performance of your device through the Edge Impulse studio

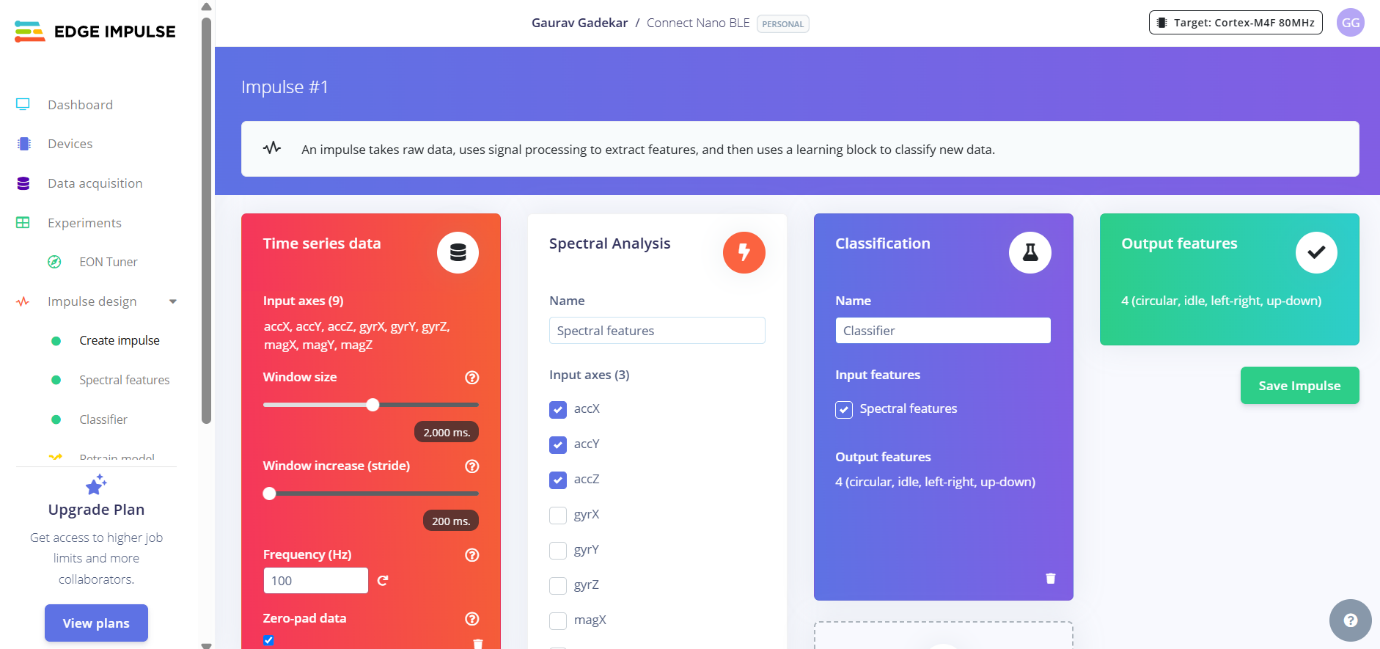
**Conclusion:** Understood Accelerometer Motion "sensor reading equivalent in Edge Impulse would typically involve creating a simple machine learning model with low computing speed and power.

**Screenshots:**

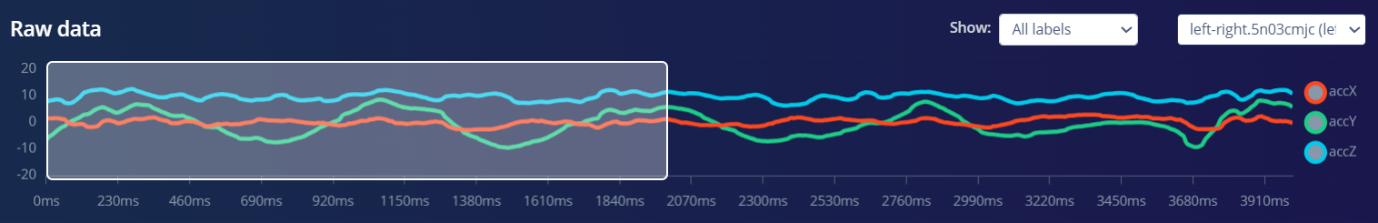
1. Dataset Image

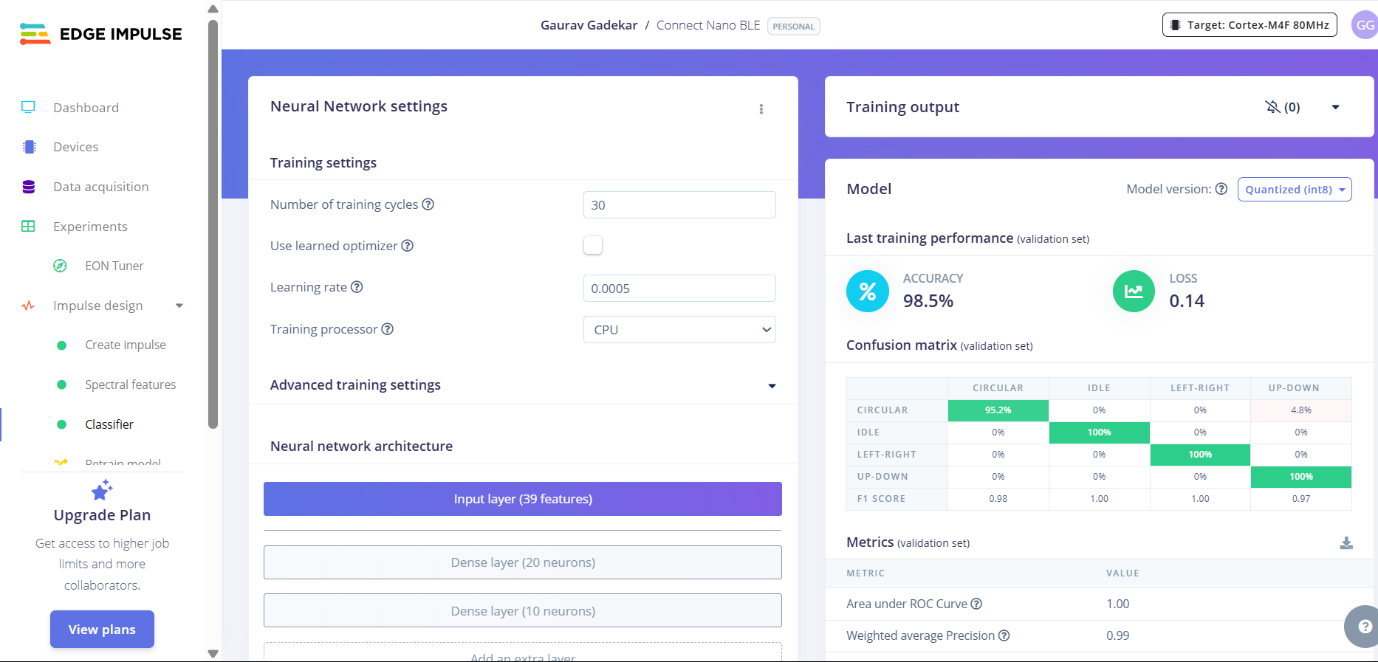


1. Feature extraction - Image

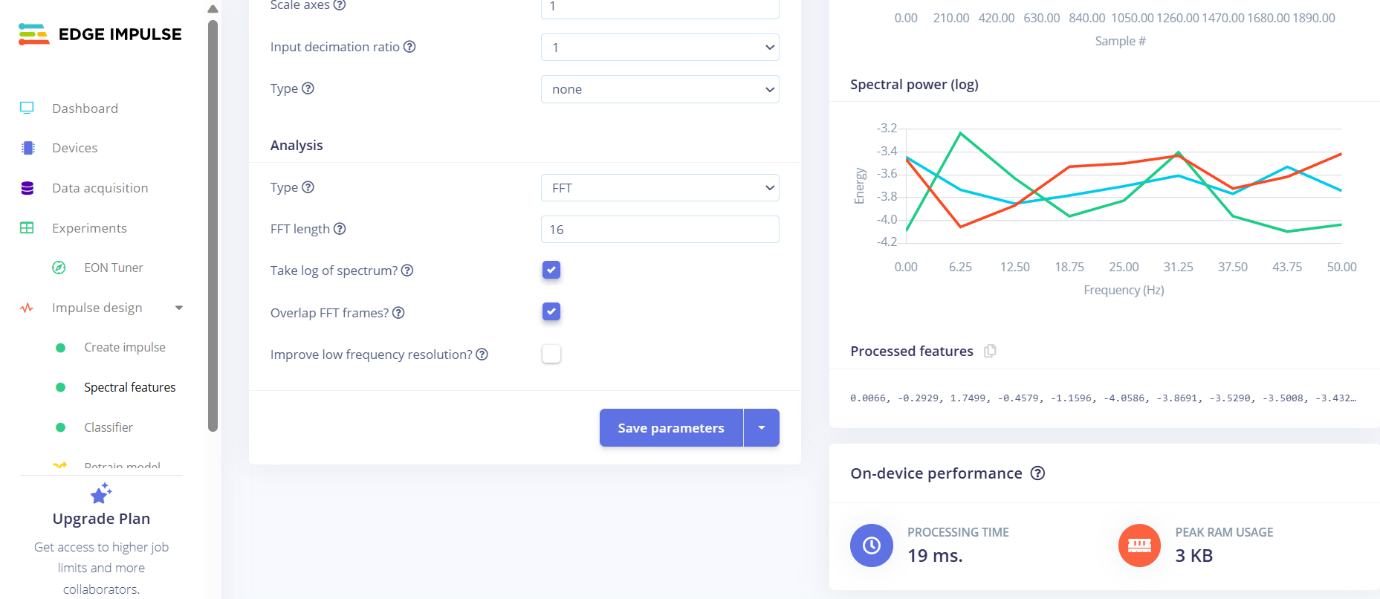


1. Accuracy / Loss - Confusion Matrix – image





# Validation Result – Image



# **Copy the code of Arduino Sketch**

/\* Edge Impulse ingestion SDK

\* Copyright (c) 2022 EdgeImpulse Inc.

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*

\*/

/\* Includes ---------------------------------------------------------------- \*/

#include <Connect\_Nano\_BLE\_inferencing.h>

#include <Arduino\_LSM9DS1.h> //Click here to get the library: https://www.arduino.cc/reference/en/libraries/arduino\_lsm9ds1/

/\* Constant defines -------------------------------------------------------- \*/

#define CONVERT\_G\_TO\_MS2 9.80665f

/\*\*

\* When data is collected by the Edge Impulse Arduino Nano 33 BLE Sense

\* firmware, it is limited to a 2G range. If the model was created with a

\* different sample range, modify this constant to match the input values.

\* See https://github.com/edgeimpulse/firmware-arduino-nano-33-ble-sense/blob/master/src/sensors/ei\_lsm9ds1.cpp

\* for more information.

\*/

#define MAX\_ACCEPTED\_RANGE 2.0f

/\*

\*\* NOTE: If you run into TFLite arena allocation issue.

\*\*

\*\* This may be due to may dynamic memory fragmentation.

\*\* Try defining "-DEI\_CLASSIFIER\_ALLOCATION\_STATIC" in boards.local.txt (create

\*\* if it doesn't exist) and copy this file to

\*\* `<ARDUINO\_CORE\_INSTALL\_PATH>/arduino/hardware/<mbed\_core>/<core\_version>/`.

\*\*

\*\* See

\*\* (https://support.arduino.cc/hc/en-us/articles/360012076960-Where-are-the-installed-cores-located-)

\*\* to find where Arduino installs cores on your machine.

\*\*

\*\* If the problem persists then there's not enough memory for this model and application.

\*/

/\* Private variables ------------------------------------------------------- \*/

static bool debug\_nn = false; // Set this to true to see e.g. features generated from the raw signal

static uint32\_t run\_inference\_every\_ms = 200;

static rtos::Thread inference\_thread(osPriorityLow);

static float buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE] = { 0 };

static float inference\_buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE];

/\* Forward declaration \*/

void run\_inference\_background();

/\*\*

\* @brief Arduino setup function

\*/

void setup()

{

// Initialize pins as outputs

pinMode(LEDR, OUTPUT);

pinMode(LEDG, OUTPUT);

pinMode(LEDB, OUTPUT);

// put your setup code here, to run once:

Serial.begin(115200);

// comment out the below line to cancel the wait for USB connection (needed for native USB)

while (!Serial);

Serial.println("Edge Impulse Inferencing Demo");

if (!IMU.begin()) {

ei\_printf("Failed to initialize IMU!\r\n");

}

else {

ei\_printf("IMU initialized\r\n");

}

if (EI\_CLASSIFIER\_RAW\_SAMPLES\_PER\_FRAME != 3) {

ei\_printf("ERR: EI\_CLASSIFIER\_RAW\_SAMPLES\_PER\_FRAME should be equal to 3 (the 3 sensor axes)\n");

return;

}

inference\_thread.start(mbed::callback(&run\_inference\_background));

}

/\*\*

\* @brief Return the sign of the number

\*

\* @param number

\* @return int 1 if positive (or 0) -1 if negative

\*/

float ei\_get\_sign(float number) {

return (number >= 0.0) ? 1.0 : -1.0;

}

/\*\*

\* @brief Run inferencing in the background.

\*/

void run\_inference\_background()

{

// wait until we have a full buffer

delay((EI\_CLASSIFIER\_INTERVAL\_MS \* EI\_CLASSIFIER\_RAW\_SAMPLE\_COUNT) + 100);

// This is a structure that smoothens the output result

// With the default settings 70% of readings should be the same before classifying.

ei\_classifier\_smooth\_t smooth;

ei\_classifier\_smooth\_init(&smooth, 10 /\* no. of readings \*/, 7 /\* min. readings the same \*/, 0.8 /\* min. confidence \*/, 0.3 /\* max anomaly \*/);

while (1) {

// copy the buffer

memcpy(inference\_buffer, buffer, EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE \* sizeof(float));

// Turn the raw buffer in a signal which we can the classify

signal\_t signal;

int err = numpy::signal\_from\_buffer(inference\_buffer, EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE, &signal);

if (err != 0) {

ei\_printf("Failed to create signal from buffer (%d)\n", err);

return;

}

// Run the classifier

ei\_impulse\_result\_t result = { 0 };

err = run\_classifier(&signal, &result, debug\_nn);

if (err != EI\_IMPULSE\_OK) {

ei\_printf("ERR: Failed to run classifier (%d)\n", err);

return;

}

// print the predictions

ei\_printf("Predictions ");

ei\_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",

result.timing.dsp, result.timing.classification, result.timing.anomaly);

ei\_printf(": ");

// ei\_classifier\_smooth\_update yields the predicted label

const char \*prediction = ei\_classifier\_smooth\_update(&smooth, &result);

ei\_printf("%s ", prediction);

// print the cumulative results

ei\_printf(" [ ");

for (size\_t ix = 0; ix < smooth.count\_size; ix++) {

ei\_printf("%u", smooth.count[ix]);

if (ix != smooth.count\_size + 1) {

ei\_printf(", ");

}

else {

ei\_printf(" ");

}

}

ei\_printf("]\n");

delay(run\_inference\_every\_ms);

}

ei\_classifier\_smooth\_free(&smooth);

}

/\*\*

\* @brief Get data and run inferencing

\*

\* @param[in] debug Get debug info if true

\*/

void loop()

{

while (1) {

// Determine the next tick (and then sleep later)

uint64\_t next\_tick = micros() + (EI\_CLASSIFIER\_INTERVAL\_MS \* 1000);

// roll the buffer -3 points so we can overwrite the last one

numpy::roll(buffer, EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE, -3);

// read to the end of the buffer

IMU.readAcceleration(

buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE - 3],

buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE - 2],

buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE - 1]

);

for (int i = 0; i < 3; i++) {

if (fabs(buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE - 3 + i]) > MAX\_ACCEPTED\_RANGE) {

buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE - 3 + i] = ei\_get\_sign(buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE - 3 + i]) \* MAX\_ACCEPTED\_RANGE;

}

}

buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE - 3] \*= CONVERT\_G\_TO\_MS2;

buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE - 2] \*= CONVERT\_G\_TO\_MS2;

buffer[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE - 1] \*= CONVERT\_G\_TO\_MS2;

// and wait for next tick

uint64\_t time\_to\_wait = next\_tick - micros();

delay((int)floor((float)time\_to\_wait / 1000.0f));

delayMicroseconds(time\_to\_wait % 1000);

char prediction;

if (prediction == "idle")

{

// WHITE

digitalWrite(LEDR, LOW);

digitalWrite(LEDG, LOW);

digitalWrite(LEDB, LOW);

}

else if (prediction == "left-right"){

// RED

digitalWrite(LEDR, LOW);

digitalWrite(LEDG, HIGH);

digitalWrite(LEDB, HIGH);

// wait for a second

delay(1000);

}

else if (prediction == "up-down"){

// GREEN

digitalWrite(LEDR, HIGH);

digitalWrite(LEDG, LOW);

digitalWrite(LEDB, HIGH);

// wait for a second

delay(1000);

}

else if (prediction == "circular"){

// BLUE

digitalWrite(LEDR, HIGH);

digitalWrite(LEDG, HIGH);

digitalWrite(LEDB, LOW);

// wait for a second

delay(1000);

}

}

}

#if !defined(EI\_CLASSIFIER\_SENSOR) || EI\_CLASSIFIER\_SENSOR != EI\_CLASSIFIER\_SENSOR\_ACCELEROMETER

#error "Invalid model for current sensor"

#endif

# Screen shot of Arduino Terminal – Result

