Edge Computing Lab

Class: TY-AIEC

School of Computing, MIT Art Design Technology University

Academic Year: 2024-25

Experiment No. 8

TASHU DHOTE

2223307

TY AIEC

Introduction

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:

1. Create an Account and New Project:
    - Sign up for an Edge Impulse account.

- Create a new project from the dashboard.

2. 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.

3. 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.

4. 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.

5. 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.

6. Train the Model:

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

7. Test the Model:

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

8. 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.

9. Run Inference:

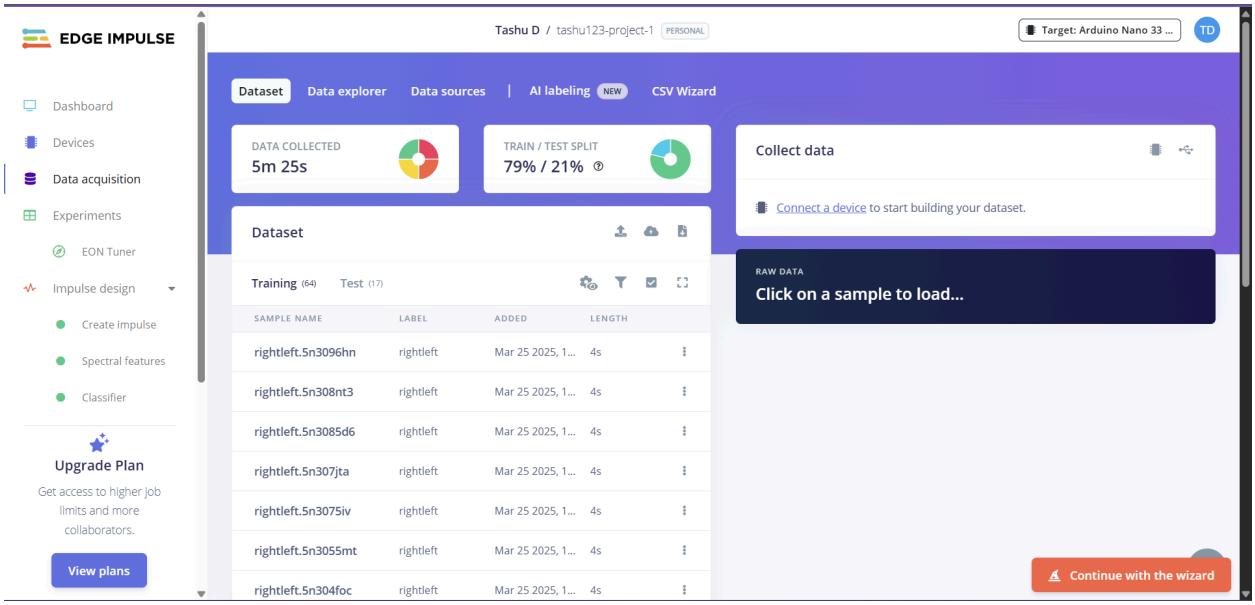
- With the model deployed, run inference on the edge device to see it classifying data in real-time.

10. Monitor:

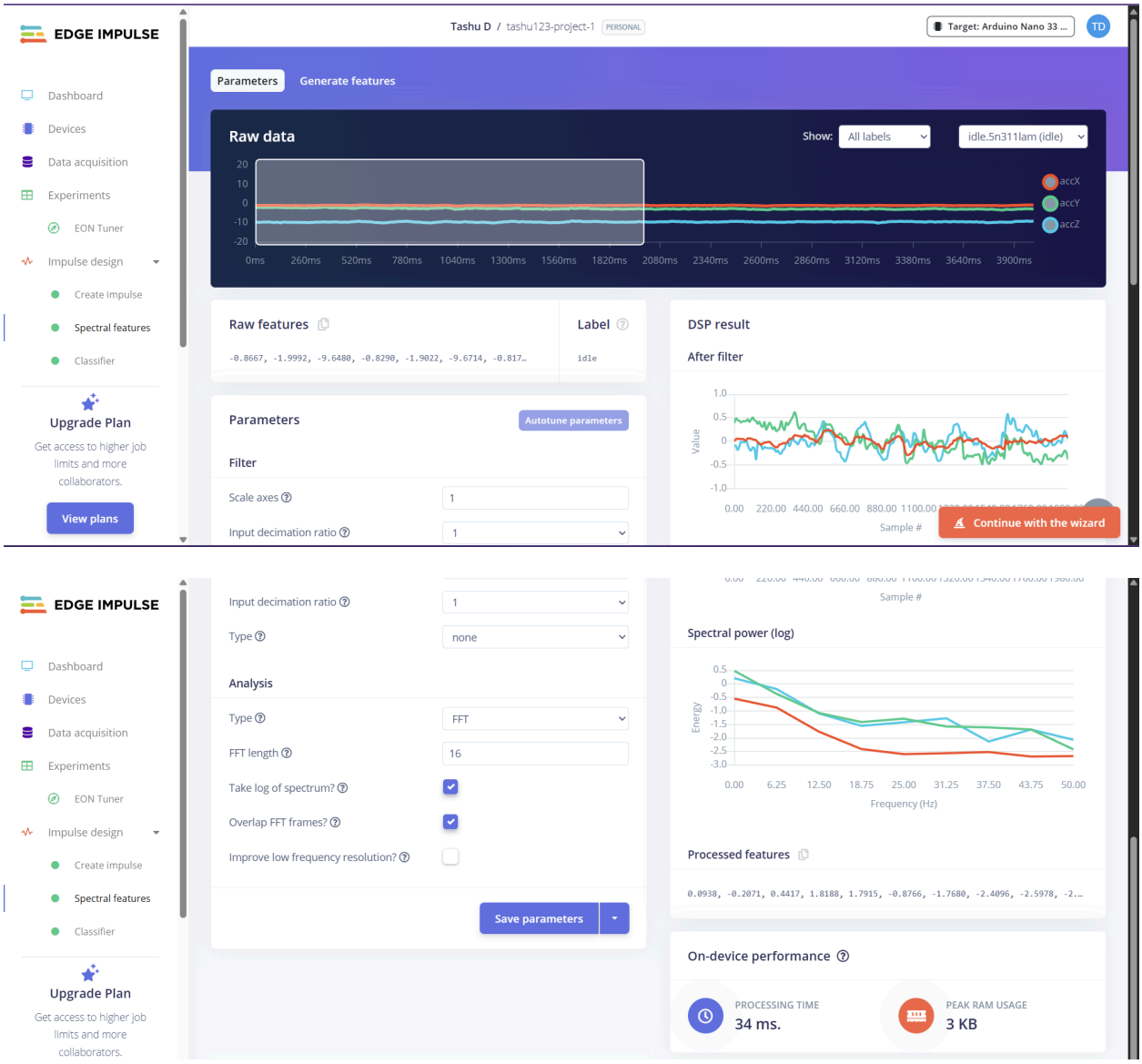
- You can monitor the performance of your device through the Edge Impulse studio.

Paste your Edge Impulse project's Results:

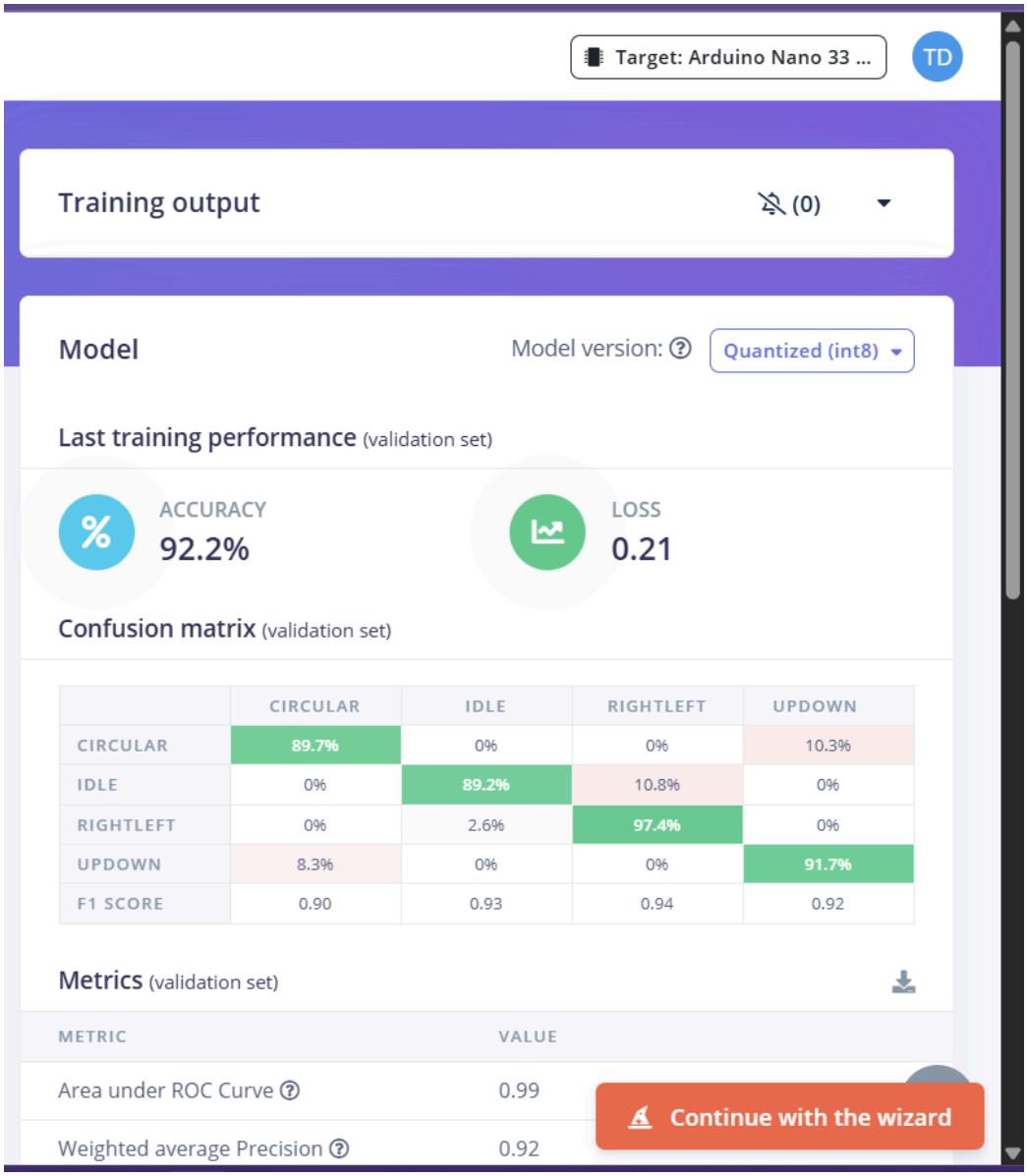
1) Dataset Image

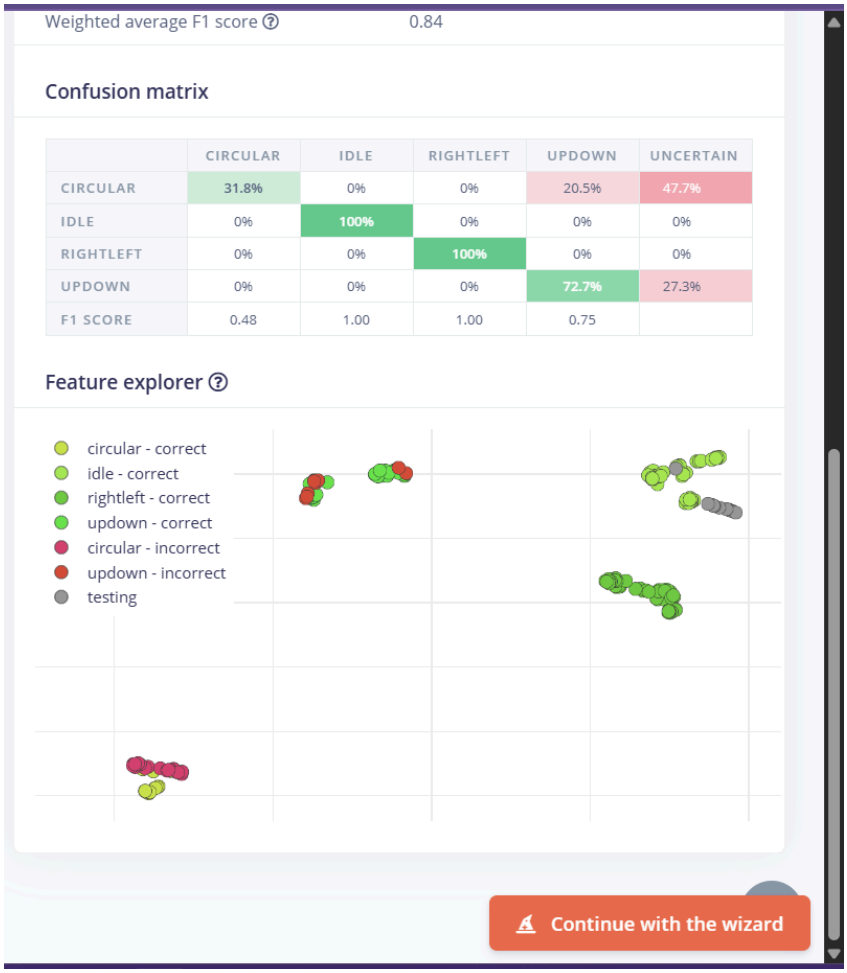


2) Feature extraction - Image

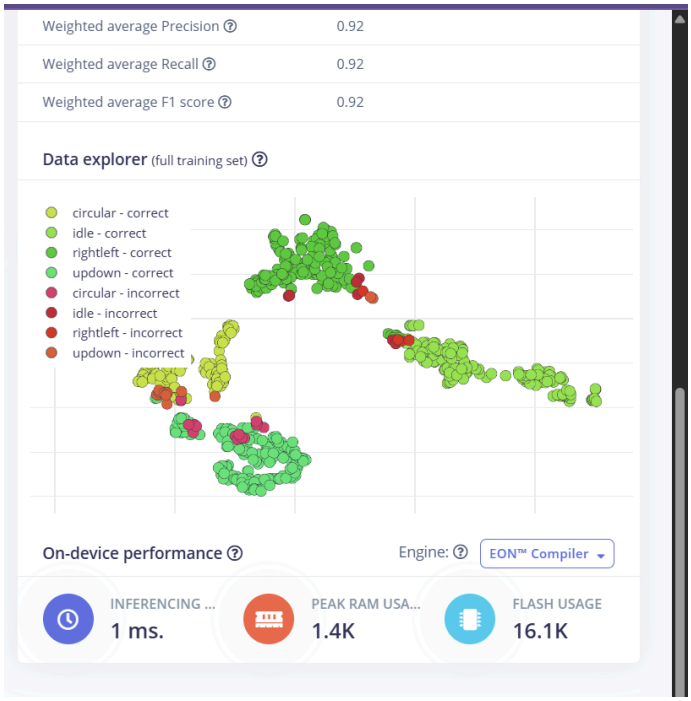


3) Accuracy / Loss - Confusion Matrix – image


Target: Arduino Nano 33 ...
TD
Training output
(0)
Model
Model version:
Quantized (int8)
Last training performance (validation set)
ACCURACY
92.2%
LOSS
0.21
Confusion matrix (validation set)
CIRCULAR IDLE RIGHTLEFT UPDOWN
CIRCULAR 89.7% 0% 0% 10.3%
IDLE 0% 89.2% 10.8% 0%
RIGHTLEFT 0% 2.6% 97.4% 0%
UPDOWN 8.3% 0% 0% 91.7%
F1 SCORE 0.90 0.93 0.94 0.92
Metrics (validation set)
METRIC VALUE
Area under ROC Curve 0.99
Weighted average Precision 0.92
Continue with the wizard



4) Validation Result – Image



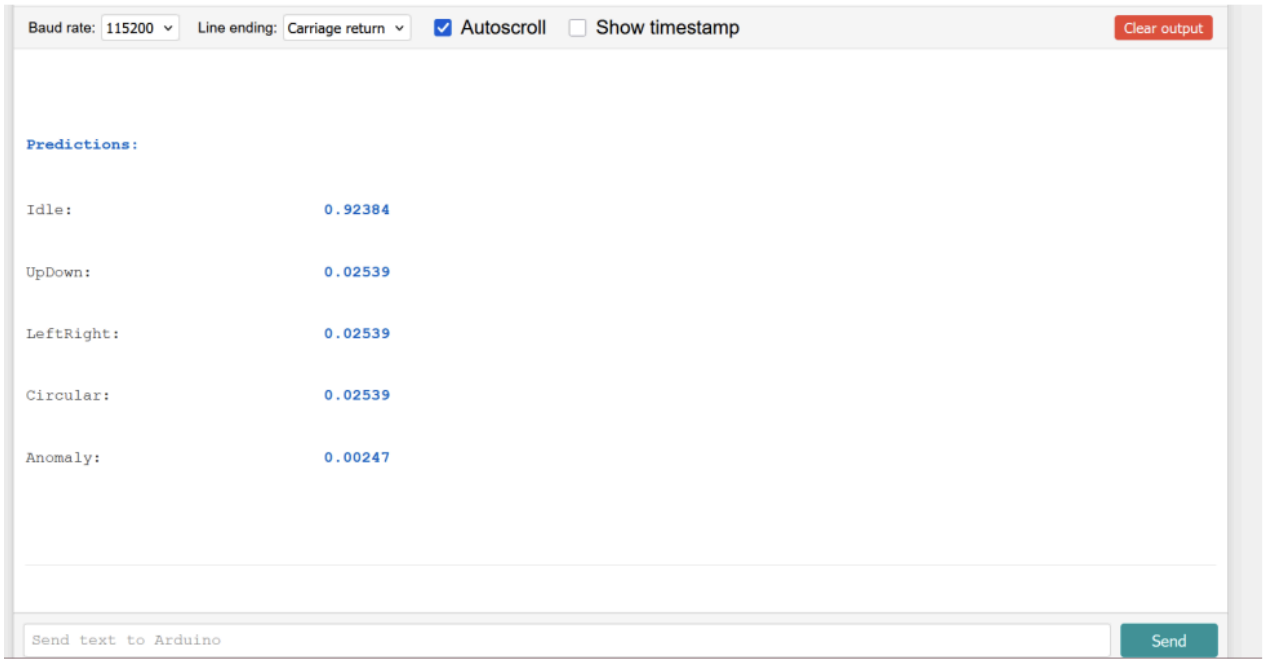
5) Copy the code of Arduino Sketch

```
#include <tashu-project-1_inferencing.h>
#include <Arduino_LSM9DS1.h>

#define CONVERT_G_TO_MS2    9.80665f

#define MAX_ACCEPTED_RANGE  2.0f

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()
{
    // 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);

    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;
```

```
58  {
65      while (1) {
69          // Turn the raw buffer in a signal which we can the classify
70          signal_t signal;
71          int err = numpy::signal_from_buffer(inference_buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
72          if (err != 0) {
73              ei_printf("Failed to create signal from buffer (%d)\n", err);
74              return;
75          }
76
77          // Run the classifier
78          ei_impulse_result_t result = { 0 };
79
80          err = run_classifier(&signal, &result, debug_nn);
81          if (err != EI_IMPULSE_OK) {
82              ei_printf("ERR: Failed to run classifier (%d)\n", err);
83              return;
84          }
85
86          // print the predictions
87          ei_printf("Predictions ");
88          ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
89              result.timing.dsp, result.timing.classification, result.timing.anomaly);
90          ei_printf(": ");
91
92          // ei_classifier_smooth_update yields the predicted label
93          const char *prediction = ei_classifier_smooth_update(&smooth, &result);
94          ei_printf("%s ", prediction);
95          // print the cumulative results
96          ei_printf(" [ ");
97          for (size_t ix = 0; ix < smooth.count_size; ix++) {
98              ei_printf("%u", smooth.count[ix]);
99              if (ix != smooth.count_size + 1) {
100                 ei_printf(", ");
101             }
```

```
120 {
121     while (1) {
124
125         // roll the buffer -3 points so we can overwrite the last one
126         numpy::roll(buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, -3);
127
128         // read to the end of the buffer
129         IMU.readAcceleration(
130             buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3],
131             buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 2],
132             buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 1]
133         );
134
135         for (int i = 0; i < 3; i++) {
136             if (fabs(buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3 + i]) > MAX_ACCEPTED_RANGE) {
137                 buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3 + i] = ei_get_sign(buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3 + i]) * MAX_ACCEPTED_RANGE;
138             }
139         }
140
141         buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3] *= CONVERT_G_TO_MS2;
142         buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 2] *= CONVERT_G_TO_MS2;
143         buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 1] *= CONVERT_G_TO_MS2;
144
145         // and wait for next tick
146         uint64_t time_to_wait = next_tick - micros();
147         delay((int)floor((float)time_to_wait / 1000.0f));
148         delayMicroseconds(time_to_wait % 1000);
149     }
150 }
151
152 #if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR != EI_CLASSIFIER_SENSOR_ACCELEROMETER
153 #error "Invalid model for current sensor"
154 #endif
```

6) Screen shot of Arduino Terminal - Result


Baud rate: 115200
Line ending: Carriage return
Autoscroll
Show timestamp
Clear output
Predictions:
Idle: 0.92384
UpDown: 0.02539
LeftRight: 0.02539
Circular: 0.02539
Anomaly: 0.00247
Send text to Arduino
Send