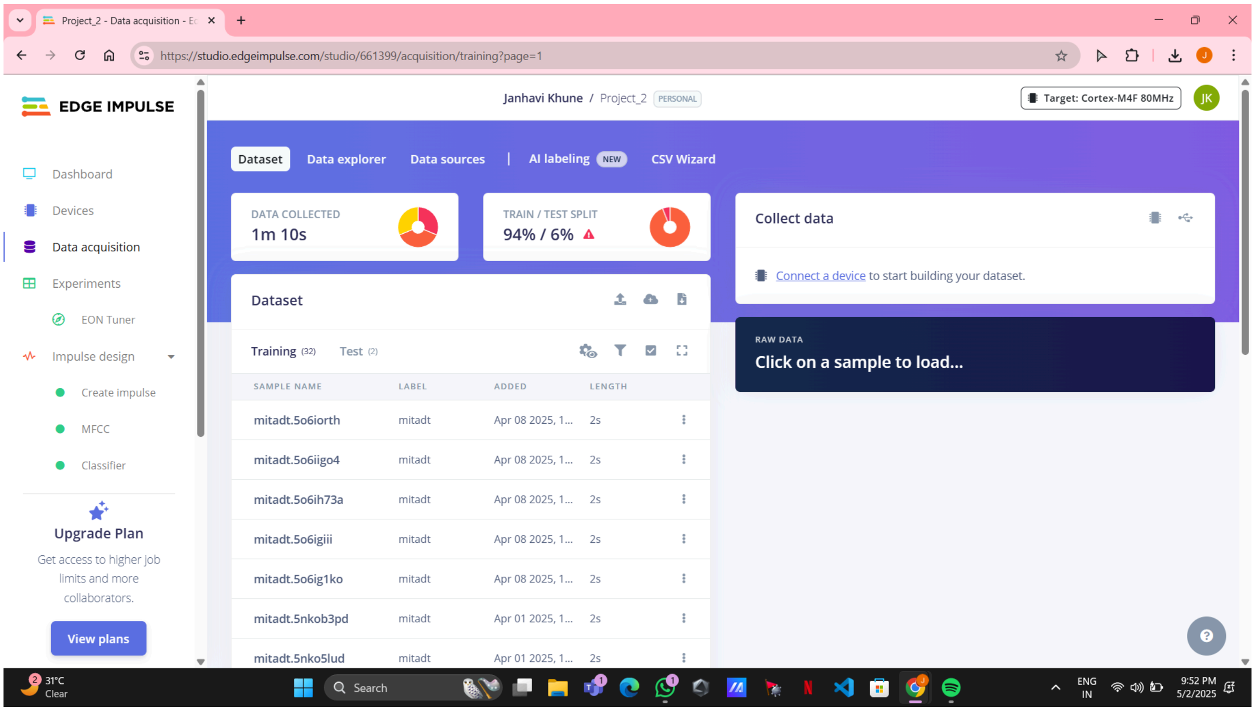
Name : Janhavi Avinash Khune
Class: TY-15 (AIEC-1)
Roll no. 222S3862
Subject: ECL

Experiment 7

Dataset training -



DATA COLLECTED
1m 10s

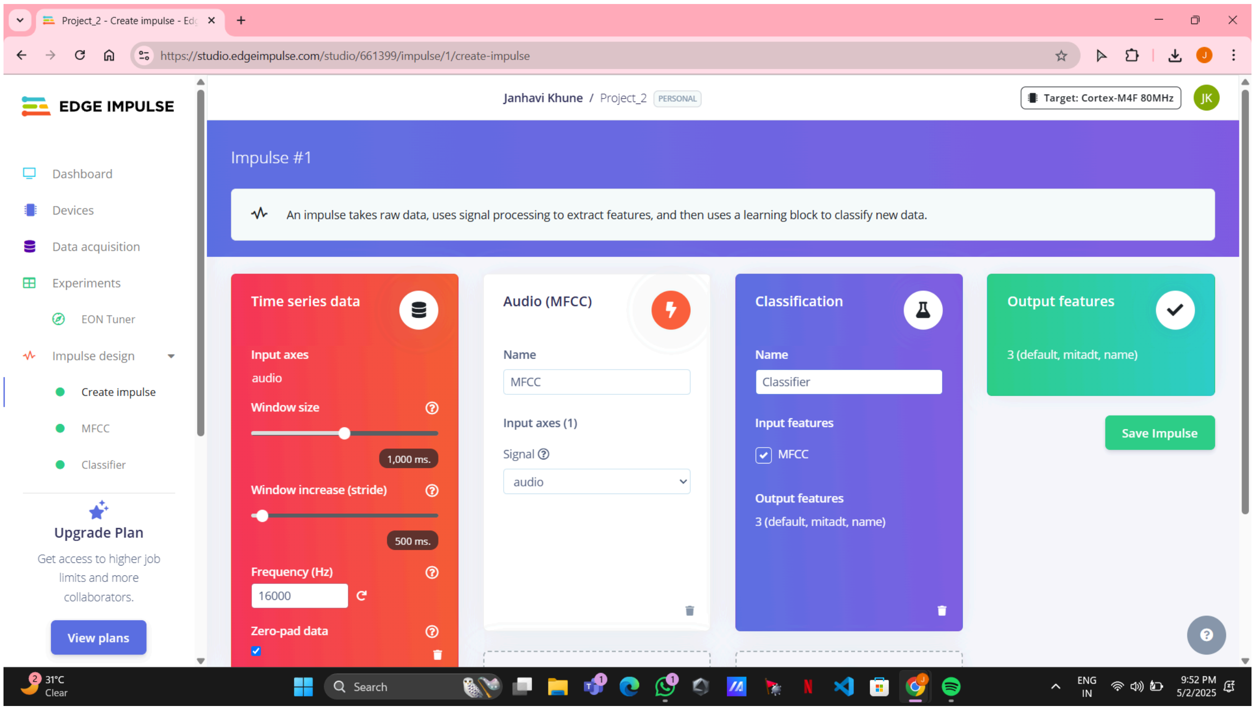
TRAIN / TEST SPLIT
94% / 6%

Connect a device to start building your dataset.

Dataset

Training (32) Test (2)

SAMPLE NAME	LABEL	ADDED	LENGTH
mitadt.5o6iorth	mitadt	Apr 08 2025, 1...	2s
mitadt.5o6iigo4	mitadt	Apr 08 2025, 1...	2s
mitadt.5o6ih73a	mitadt	Apr 08 2025, 1...	2s
mitadt.5o6igiii	mitadt	Apr 08 2025, 1...	2s
mitadt.5o6ig1ko	mitadt	Apr 08 2025, 1...	2s
mitadt.5nkob3pd	mitadt	Apr 01 2025, 1...	2s
mitadt.5nko5lud	mitadt	Apr 01 2025, 1...	2s



Project_2 - Create impulse - Ed
https://studio.edgeimpulse.com/studio/661399/impulse/1/create-impulse
EDGE IMPULSE
Dashboard
Devices
Data acquisition
Experiments
EON Tuner
Impulse design
Create impulse
MFCC
Classifier
Upgrade Plan
Get access to higher job limits and more collaborators.
View plans
Janhavi Khune / Project_2 PERSONAL
Target: Cortex-M4F 80MHz
JK

Impulse #1

An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.

Time series data

Input axes: audio

Window size: 1,000 ms.

Window increase (stride): 500 ms.

Frequency (Hz): 16000

Zero-pad data ☑

Audio (MFCC)

Name: MFCC

Input axes (1)

Signal: audio

Classification

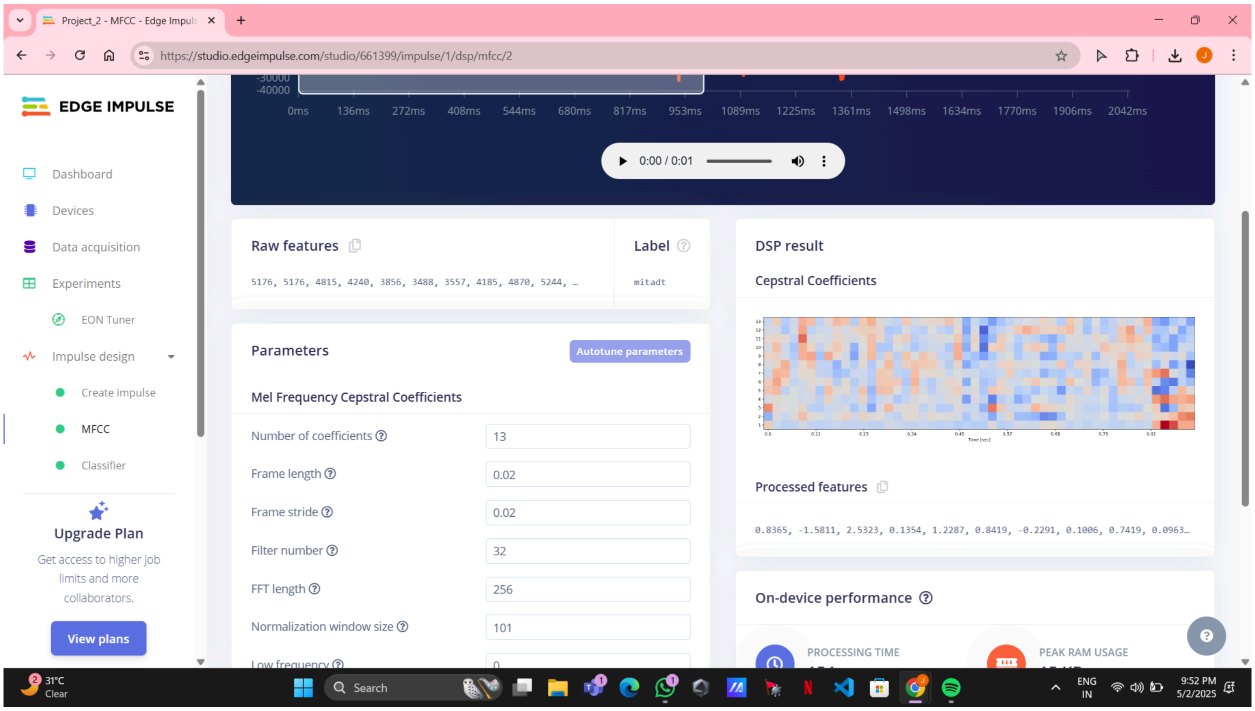
Name: Classifier

Input features: ☑ MFCC

Output features: 3 (default, mitadt, name)

Output features

3 (default, mitadt, name)



Raw features

5176, 5176, 4815, 4240, 3856, 3488, 3557, 4185, 4870, 5244, …

Label

mitadt

Parameters

Mel Frequency Cepstral Coefficients

Number of coefficients: 13

Frame length: 0.02

Frame stride: 0.02

Filter number: 32

FFT length: 256

Normalization window size: 101

Low frequency: 0

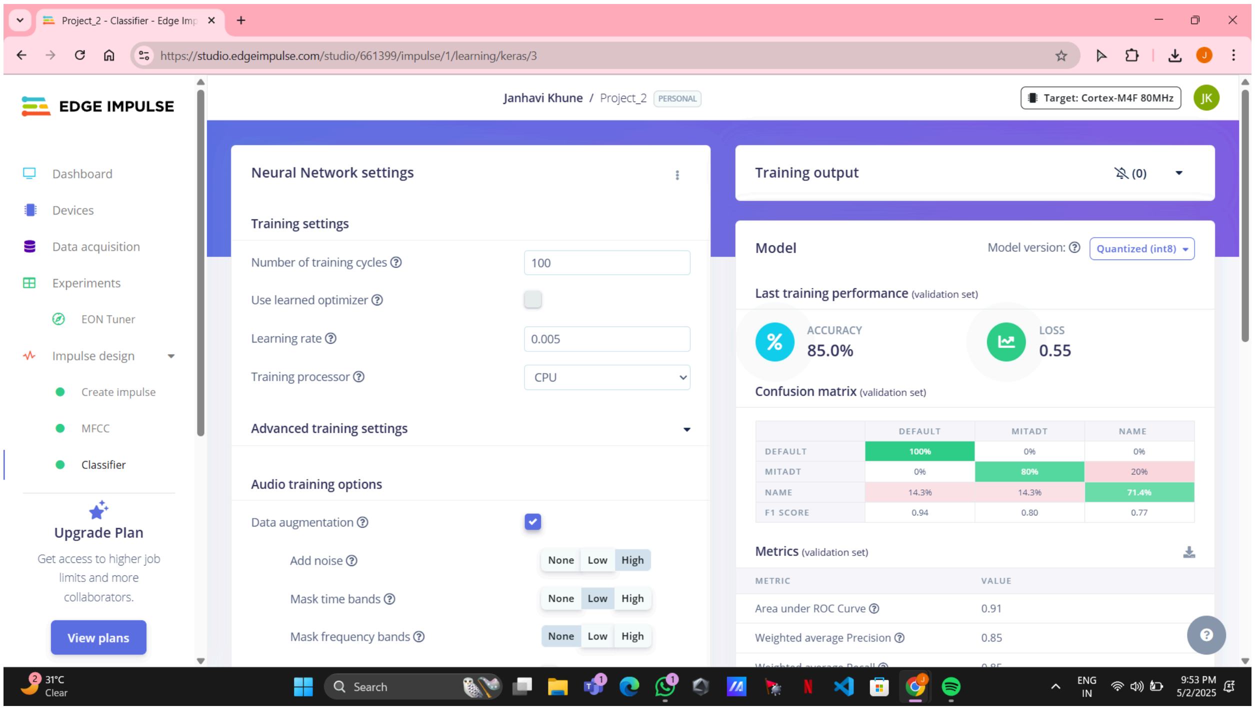
DSP result

Cepstral Coefficients

Processed features

0.8365, -1.5811, 2.5323, 0.1354, 1.2287, 0.8419, -0.2291, 0.1006, 0.7419, 0.0963…

On-device performance


Project_2 - Classifier - Edge Imp
https://studio.edgeimpulse.com/studio/661399/impulse/1/learning/keras/3
EDGE IMPULSE
Dashboard
Devices
Data acquisition
Experiments
EON Tuner
Impulse design
Create impulse
MFCC
Classifier
Upgrade Plan
Get access to higher job limits and more collaborators.
View plans
Janhavi Khune / Project_2 PERSONAL
Target: Cortex-M4F 80MHz
JK
31°C
Clear
Search
ENG
IN
9:53 PM
5/2/2025

Neural Network settings

Training settings

Number of training cycles	100
Use learned optimizer	□
Learning rate	0.005
Training processor	CPU

Advanced training settings

Audio training options

Data augmentation	☑
Add noise	None Low [✓ High]
Mask time bands	None [✓ Low] High
Mask frequency bands	[✓ None] Low High

Training output

(0)

Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 85.0%

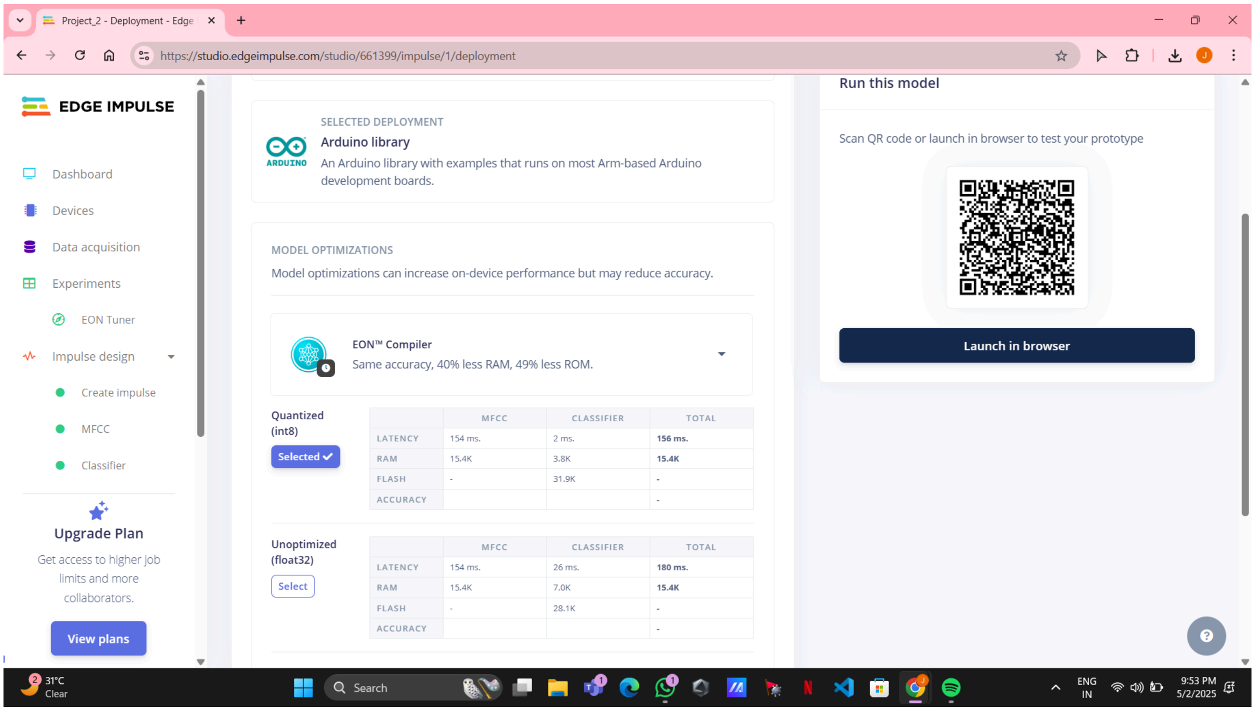
LOSS 0.55

Confusion matrix (validation set)

	DEFAULT	MITADT	NAME
DEFAULT	100%	0%	0%
MITADT	0%	80%	20%
NAME	14.3%	14.3%	71.4%
F1 SCORE	0.94	0.80	0.77

Metrics (validation set)

METRIC	VALUE
Area under ROC Curve	0.91
Weighted average Precision	0.85
Weighted average Recall	0.85



SELECTED DEPLOYMENT

Arduino library

An Arduino library with examples that runs on most Arm-based Arduino development boards.

MODEL OPTIMIZATIONS

Model optimizations can increase on-device performance but may reduce accuracy.

EON™ Compiler

Same accuracy, 40% less RAM, 49% less ROM.

Quantized (int8) — Selected ✓

	MFCC	CLASSIFIER	TOTAL
LATENCY	154 ms.	2 ms.	**156 ms.**
RAM	15.4K	3.8K	**15.4K**
FLASH	-	31.9K	-
ACCURACY			-

Unoptimized (float32) — Select

	MFCC	CLASSIFIER	TOTAL
LATENCY	154 ms.	26 ms.	**180 ms.**
RAM	15.4K	7.0K	**15.4K**
FLASH	-	28.1K	-
ACCURACY			-

Run this model

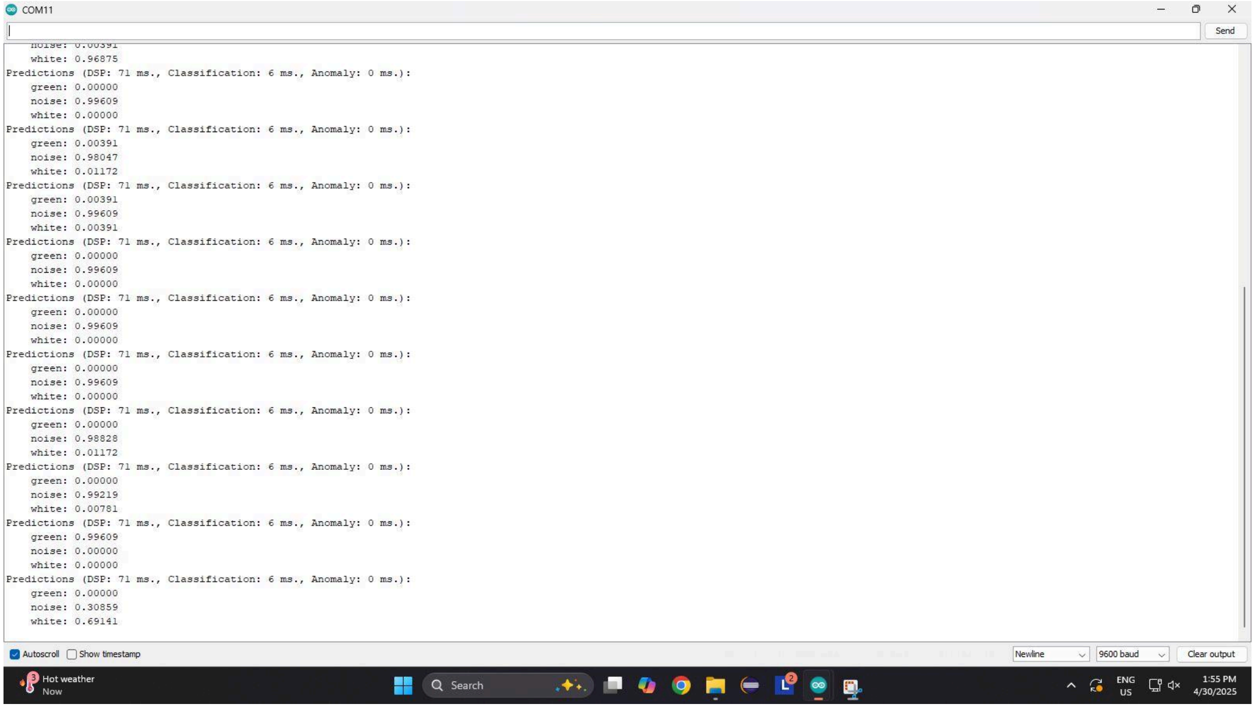
Scan QR code or launch in browser to test your prototype

Code:-

```
/* 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 <Project_2_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/mas
ter/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-i
nstalled-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()
{
    // 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);
    }
}

#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR !=
EI_CLASSIFIER_SENSOR_ACCELEROMETER
#error "Invalid model for current sensor"
#endif
```

Output:-



```
    noise: 0.00391
    white: 0.96875
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
    green: 0.00000
    noise: 0.99609
    white: 0.00000
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
    green: 0.00391
    noise: 0.98047
    white: 0.01172
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
    green: 0.00391
    noise: 0.99609
    white: 0.00391
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
    green: 0.00000
    noise: 0.99609
    white: 0.00000
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
    green: 0.00000
    noise: 0.99609
    white: 0.00000
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
    green: 0.00000
    noise: 0.99609
    white: 0.00000
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
    green: 0.00000
    noise: 0.98828
    white: 0.01172
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
    green: 0.00000
    noise: 0.99219
    white: 0.00781
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
    green: 0.99609
    noise: 0.00000
    white: 0.00000
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
    green: 0.00000
    noise: 0.30859
    white: 0.69141
```