Edge Computing Lab

Class: TY-AIEC

School of Computing, MIT Art Design Technology University

*Academic Year: 2024-25*

**Experiment No. 7**

Name:HARSHIT PRAKASH SHETTY
Class:TY-AIEC-B
Enrollment No: MITU22BTCS0321
Roll No: 2223617

## Introduction

Study of Classification learning block using a NN Classifier on Edge Devices

Objective: Build a project to detect the keywords using built-in sensor on Nano BLE Sense / Mobile Phone

Tasks:

- Generate the dataset for keyword
- Configure BLE Sense / Mobile for Edge Impulse
- Building and Training a Model

Study of **Confusion matrix**

## Introduction

Edge Impulse is a development platform for machine learning on edge devices, targeted at developers who want to create intelligent device solutions. The "classification block" 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:

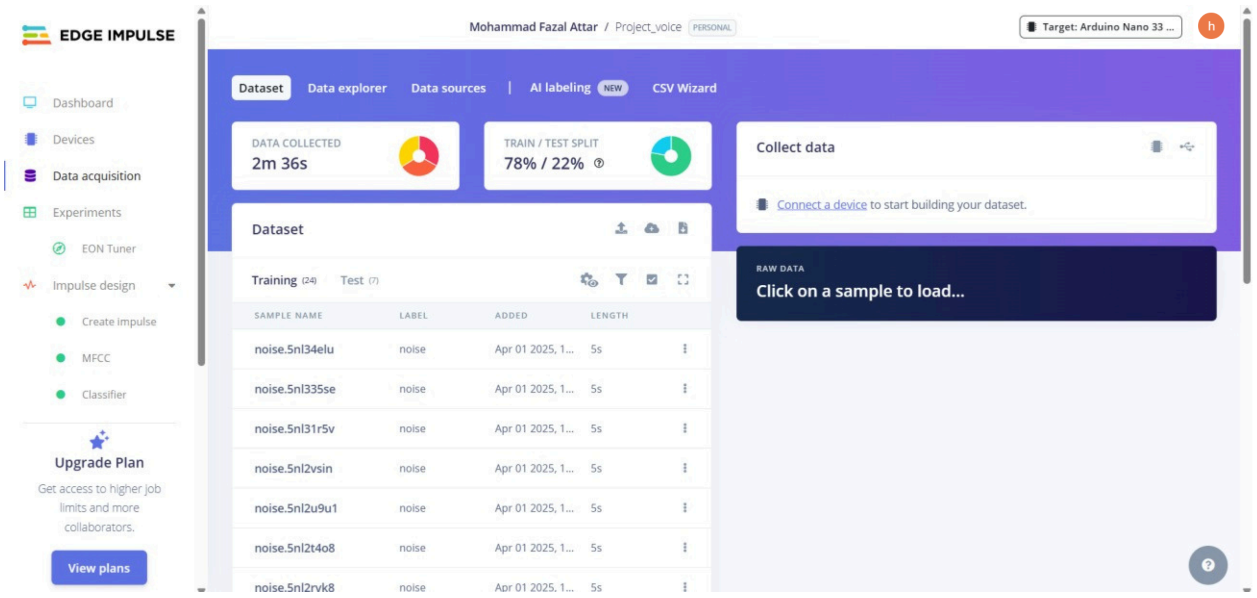
   - With the model deployed, run inference on the edge device to see it classifying data in real-time.

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

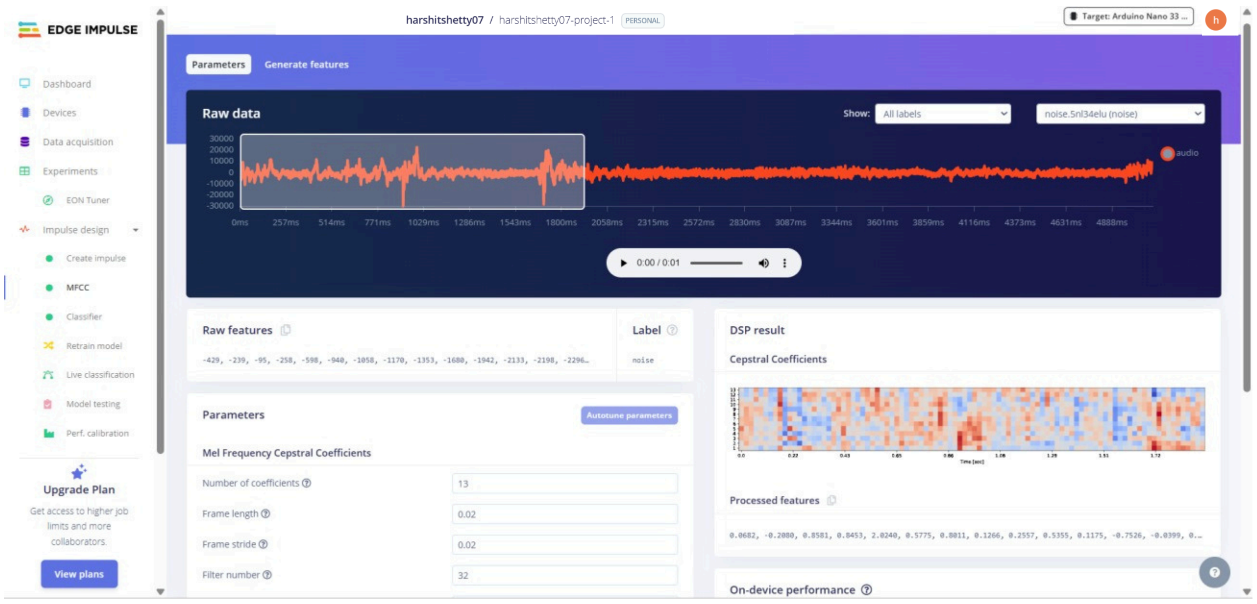


DATA COLLECTED 2m 36s

TRAIN / TEST SPLIT 78% / 22%

| SAMPLE NAME | LABEL | ADDED | LENGTH |
|---|---|---|---|
| noise.5nl34elu | noise | Apr 01 2025, 1... | 5s |
| noise.5nl335se | noise | Apr 01 2025, 1... | 5s |
| noise.5nl31r5v | noise | Apr 01 2025, 1... | 5s |
| noise.5nl2vsin | noise | Apr 01 2025, 1... | 5s |
| noise.5nl2u9u1 | noise | Apr 01 2025, 1... | 5s |
| noise.5nl2t4o8 | noise | Apr 01 2025, 1... | 5s |
| noise.5nl2rvk8 | noise | Apr 01 2025, 1... | 5s |

## 2) Feature extraction – Image



Raw features

```
-429, -239, -95, -258, -598, -940, -1058, -1170, -1353, -1680, -1942, -2133, -2198, -2296…
```

Label: noise

Parameters

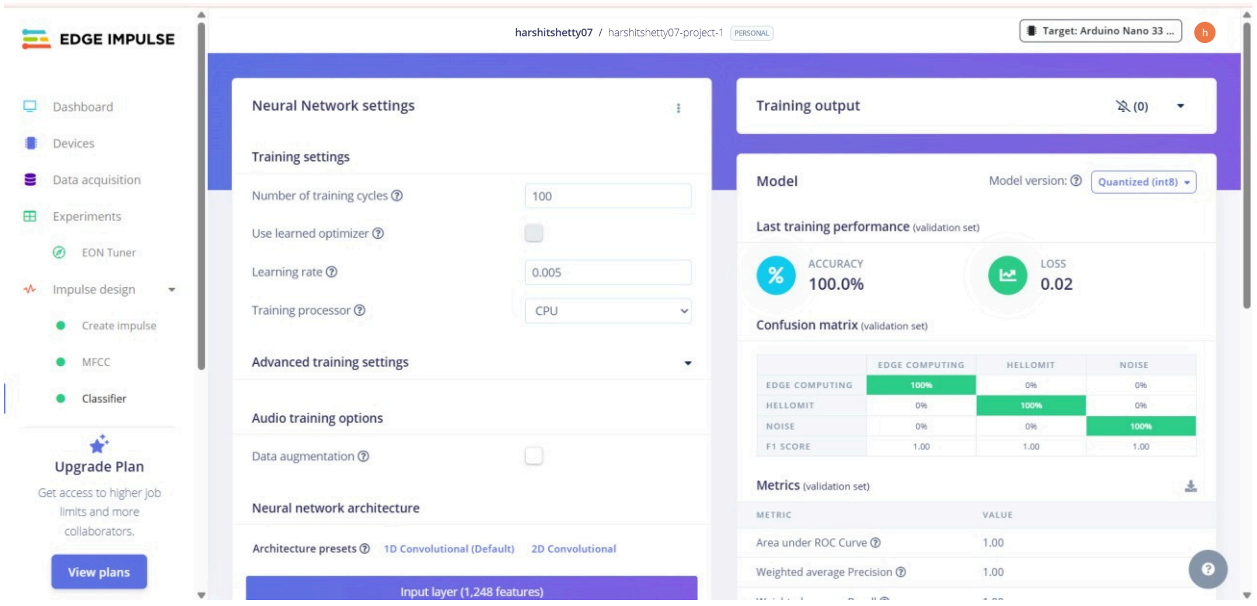
Mel Frequency Cepstral Coefficients

| | |
|---|---|
| Number of coefficients | 13 |
| Frame length | 0.02 |
| Frame stride | 0.02 |
| Filter number | 32 |

Processed features

```
0.0682, -0.2080, 0.8581, 0.8453, 2.0240, 0.5775, 0.8011, 0.1266, 0.2557, 0.5355, 0.1175, -0.7526, -0.0399, 0...
```

## 3) Accuracy / Loss - Confusion Matrix – image


harshitshetty07 / harshitshetty07-project-1 PERSONAL

Target: Arduino Nano 33 ...

**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 | ☐ |

Neural network architecture

Architecture presets: 1D Convolutional (Default) 2D Convolutional

Input layer (1,248 features)

**Training output**

**Model** Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 100.0%

LOSS 0.02

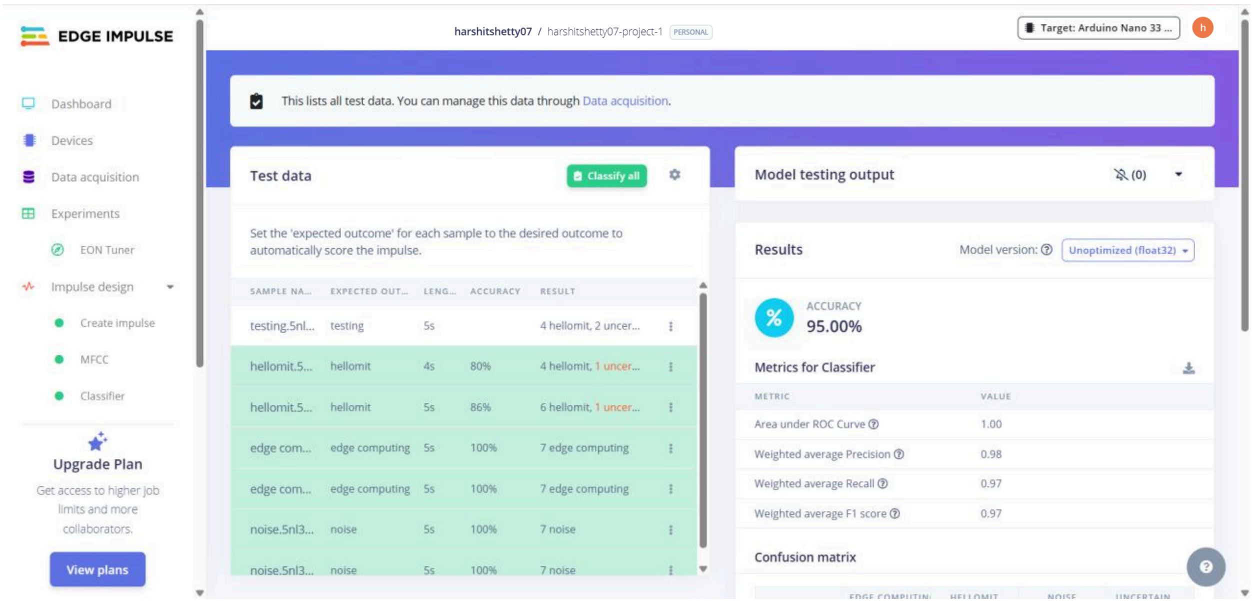
Confusion matrix (validation set)

| | EDGE COMPUTING | HELLOMIT | NOISE |
|---|---|---|---|
| EDGE COMPUTING | 100% | 0% | 0% |
| HELLOMIT | 0% | 100% | 0% |
| NOISE | 0% | 0% | 100% |
| F1 SCORE | 1.00 | 1.00 | 1.00 |

Metrics (validation set)

| METRIC | VALUE |
|---|---|
| Area under ROC Curve | 1.00 |
| Weighted average Precision | 1.00 |

## 4) Validation Result – Image


harshitshetty07 / harshitshetty07-project-1 PERSONAL

Target: Arduino Nano 33 ...

This lists all test data. You can manage this data through Data acquisition.

**Test data**

Set the 'expected outcome' for each sample to the desired outcome to automatically score the impulse.

| SAMPLE NA... | EXPECTED OUT... | LENG... | ACCURACY | RESULT |
|---|---|---|---|---|
| testing.5nl... | testing | 5s | | 4 hellomit, 2 uncer... |
| hellomit.5... | hellomit | 4s | 80% | 4 hellomit, 1 uncer... |
| hellomit.5... | hellomit | 5s | 86% | 6 hellomit, 1 uncer... |
| edge com... | edge computing | 5s | 100% | 7 edge computing |
| edge com... | edge computing | 5s | 100% | 7 edge computing |
| noise.5nl3... | noise | 5s | 100% | 7 noise |
| noise.5nl3... | noise | 5s | 100% | 7 noise |

**Model testing output**

**Results** Model version: Unoptimized (float32)

ACCURACY 95.00%

Metrics for Classifier

| METRIC | VALUE |
|---|---|
| Area under ROC Curve | 1.00 |
| Weighted average Precision | 0.98 |
| Weighted average Recall | 0.97 |
| Weighted average F1 score | 0.97 |

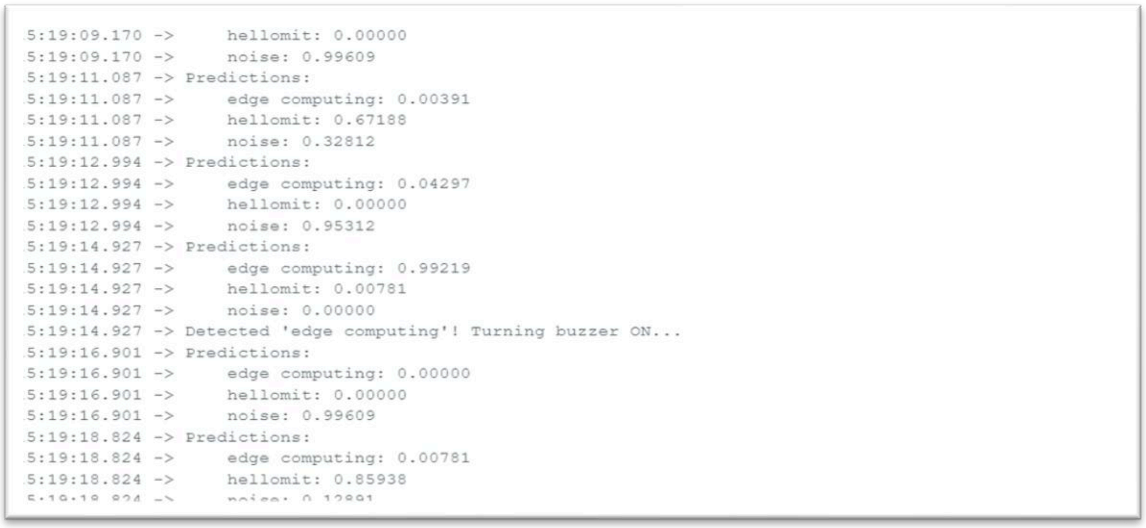
Confusion matrix

5) 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.
 *
 */


// If your target is limited in memory remove this macro to save 10K RAM
#define EIDSP_QUANTIZE_FILTERBANK   0

/**
 * Define the number of slices per model window. E.g. a model window of 1000 ms
 * with slices per model window set to 4. Results in a slice size of 250 ms.
 * For more info: https://docs.edgeimpulse.com/docs/continuous-audio-sampling
 */
#define EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW 4

/*
 ** 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.
 */

/* Includes ---------------------------------------------------------------- */
#include <PDM.h>
#include <Project_voice_inferencing.h>
```

```
/** Audio buffers, pointers and selectors */
typedef struct {
    signed  short  *buffers[2];
    unsigned  char  buf_select;
    unsigned  char   buf_ready;
    unsigned   int   buf_count;
    unsigned int n_samples;
} inference_t;

static inference_t inference;
static bool record_ready = false;
static signed short *sampleBuffer;
static bool debug_nn = false; // Set this to true to see e.g. features generated
from the raw signal
static int print_results = -(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW);

/**
 * @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");

    // summary of inferencing settings (from model_metadata.h)
    ei_printf("Inferencing settings:\n");
    ei_printf("\tInterval: %.2f ms.\n", (float)EI_CLASSIFIER_INTERVAL_MS);
    ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
    ei_printf("\tSample length: %d ms.\n", EI_CLASSIFIER_RAW_SAMPLE_COUNT / 16);
    ei_printf("\tNo. of classes: %d\n",
sizeof(ei_classifier_inferencing_categories) /
                                            sizeof(ei_classifier_inferencing_cate
gories[0]));

    run_classifier_init();
    if (microphone_inference_start(EI_CLASSIFIER_SLICE_SIZE) == false) {
        ei_printf("ERR: Could not allocate audio buffer (size %d), this could be
due to the window length of your model\r\n", EI_CLASSIFIER_RAW_SAMPLE_COUNT);
        return;
    }
}

/**
 * @brief      Arduino main function. Runs the inferencing loop.
 */
void loop()
{
    bool m = microphone_inference_record();
    if (!m) {
        ei_printf("ERR: Failed to record audio...\n");
        return;
    }

    signal_t signal;
    signal.total_length = EI_CLASSIFIER_SLICE_SIZE;
    signal.get_data = &microphone_audio_signal_get_data;
    ei_impulse_result_t result = {0};

    EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, debug_nn);
    if (r != EI_IMPULSE_OK) {
        ei_printf("ERR: Failed to run classifier (%d)\n", r);
        return;
    }

    if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)) {
        // 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(": \n");
        for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
            ei_printf("    %s: %.5f\n", result.classification[ix].label,
                      result.classification[ix].value);
        }
#if EI_CLASSIFIER_HAS_ANOMALY == 1
        ei_printf("    anomaly score: %.3f\n", result.anomaly);
#endif

        print_results = 0;
    }
}

/**
 * @brief      PDM buffer full callback
 *             Get data and call audio thread callback
 */
static void pdm_data_ready_inference_callback(void)
{
    int bytesAvailable = PDM.available();

    // read into the sample buffer
    int bytesRead = PDM.read((char *)&sampleBuffer[0], bytesAvailable);

    if (record_ready == true) {
        for (int i = 0; i<bytesRead>> 1; i++) {
            inference.buffers[inference.buf_select][inference.buf_count++] =
sampleBuffer[i];

            if (inference.buf_count >= inference.n_samples) {
                inference.buf_select ^= 1;
                inference.buf_count = 0;
                inference.buf_ready = 1;
            }
        }
    }
}

/**
 * @brief      Init inferencing struct and setup/start PDM
 *
 * @param[in]  n_samples  The n samples
 *
 * @return     { description_of_the_return_value }
 */
static bool microphone_inference_start(uint32_t n_samples)
{
    inference.buffers[0] = (signed short *)malloc(n_samples * sizeof(signed
short));

    if (inference.buffers[0] == NULL) {
        return false;
    }

    inference.buffers[1] = (signed short *)malloc(n_samples * sizeof(signed
short));

    if (inference.buffers[1] == NULL) {
    free(inference.buffers[0]);
    return false;
    }

    sampleBuffer = (signed short *)malloc((n_samples >> 1) * sizeof(signed
short));

    if (sampleBuffer == NULL) {
        free(inference.buffers[0]);
        free(inference.buffers[1]);
        return false;
    }
    inference.buf_select = 0;
    inference.buf_count = 0;
    inference.n_samples = n_samples;
    inference.buf_ready = 0;

    // configure the data receive callback
    PDM.onReceive(&pdm_data_ready_inference_callback);

    PDM.setBufferSize((n_samples >> 1) * sizeof(int16_t));

    // initialize PDM with:
    // - one channel (mono mode)
    // - a 16 kHz sample rate
    if (!PDM.begin(1, EI_CLASSIFIER_FREQUENCY)) {
        ei_printf("Failed to start PDM!");
    }

    // set the gain, defaults to 20
    PDM.setGain(127);

    record_ready = true;

    return true;
}

/**
 * @brief      Wait on new data
 *
 * @return     True when finished
 */
static bool microphone_inference_record(void)
{
    bool ret = true;

    if (inference.buf_ready == 1) {
        ei_printf(
            "Error sample buffer overrun. Decrease the number of slices per model
window "
            "(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)\n");
        ret = false;
    }

    while (inference.buf_ready == 0) {
        delay(1);
    }

    inference.buf_ready = 0;
    return ret;
}

/**
 * Get raw audio signal data
 */
static int microphone_audio_signal_get_data(size_t offset, size_t length, float
*out_ptr)
{
    numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1][offset],
out_ptr, length);

    return 0;
}

/**
 * @brief      Stop PDM and release buffers
 */
static void microphone_inference_end(void)
{
    PDM.end();
    free(inference.buffers[0]);
    free(inference.buffers[1]);
    free(sampleBuffer);
}

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

6) Screen shot of Arduino Terminal – Result



```
5:19:09.170 ->     hellomit: 0.00000
5:19:09.170 ->     noise: 0.99609
5:19:11.087 -> Predictions:
5:19:11.087 ->     edge computing: 0.00391
5:19:11.087 ->     hellomit: 0.67188
5:19:11.087 ->     noise: 0.32812
5:19:12.994 -> Predictions:
5:19:12.994 ->     edge computing: 0.04297
5:19:12.994 ->     hellomit: 0.00000
5:19:12.994 ->     noise: 0.95312
5:19:14.927 -> Predictions:
5:19:14.927 ->     edge computing: 0.99219
5:19:14.927 ->     hellomit: 0.00781
5:19:14.927 ->     noise: 0.00000
5:19:14.927 -> Detected 'edge computing'! Turning buzzer ON...
5:19:16.901 -> Predictions:
5:19:16.901 ->     edge computing: 0.00000
5:19:16.901 ->     hellomit: 0.00000
5:19:16.901 ->     noise: 0.99609
5:19:18.824 -> Predictions:
5:19:18.824 ->     edge computing: 0.00781
5:19:18.824 ->     hellomit: 0.85938
5:19:18.824 ->     noise: 0.12891
```