

Name: Tanmay Khedekar

Class: TY-15 (Batch A)

Roll Number: 2223122

Enrolment Number: MITU22BTCS0906

ECL Experiment 6

Title

Keyword Spotting Project like “OK, Google,” “Alexa,” on Edge Devices using
Microphone

Objective: Build a project to detect the keywords using a 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

Run the project Keyword Spotting like “OK, Google,” “Alexa

Introduction

Edge Impulse is a development platform for machine learning on edge devices,
targeted at
developers who want to create intelligent device solutions. The “Hello
World” 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

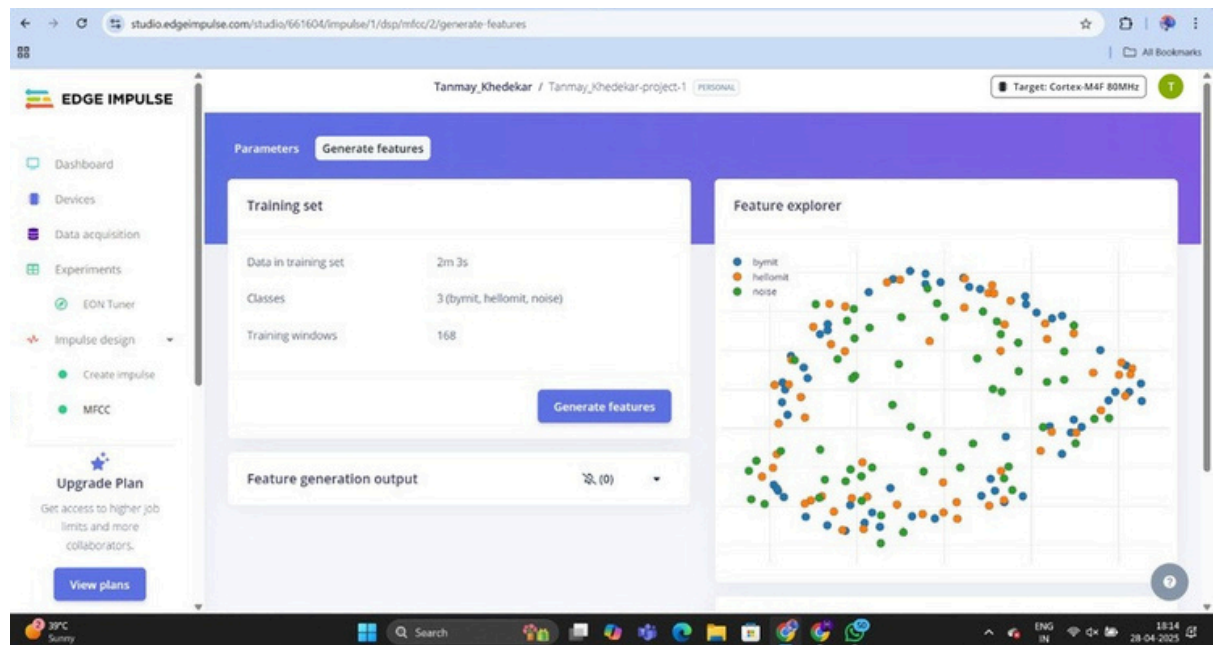
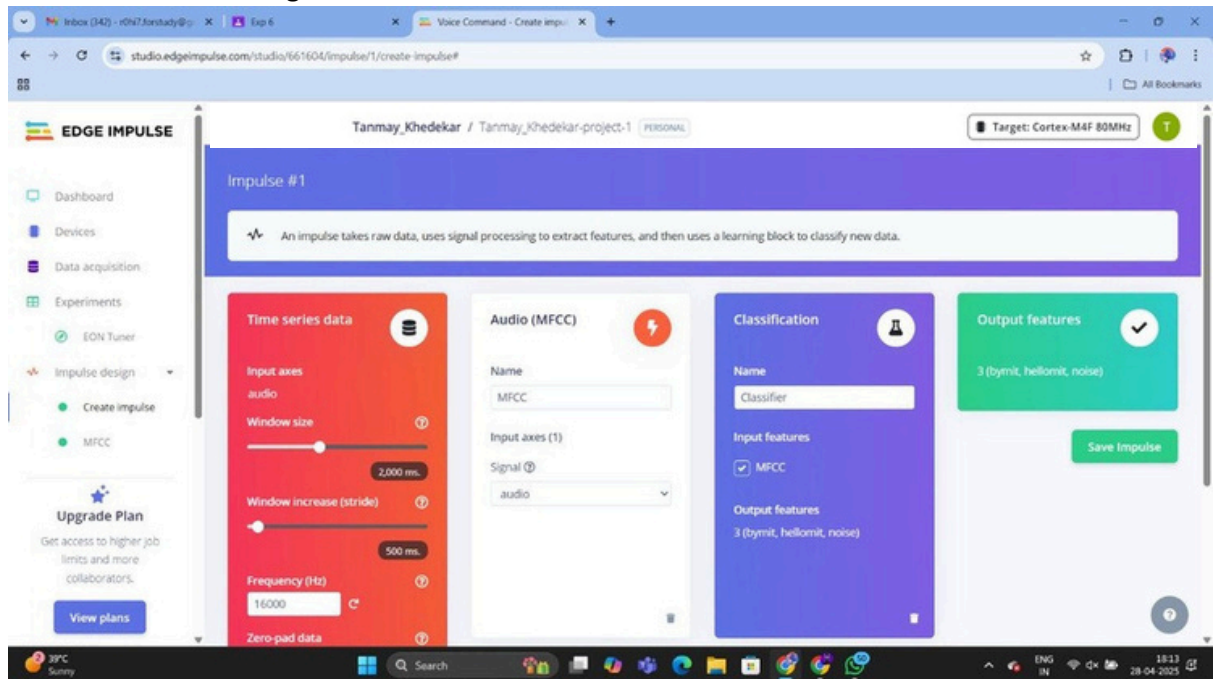
1. Dataset Image

The screenshot displays the Edge Impulse web interface for a project named 'Tanmay_Khedekar / Tanmay_Khedekar-project-1'. The interface is divided into several sections:

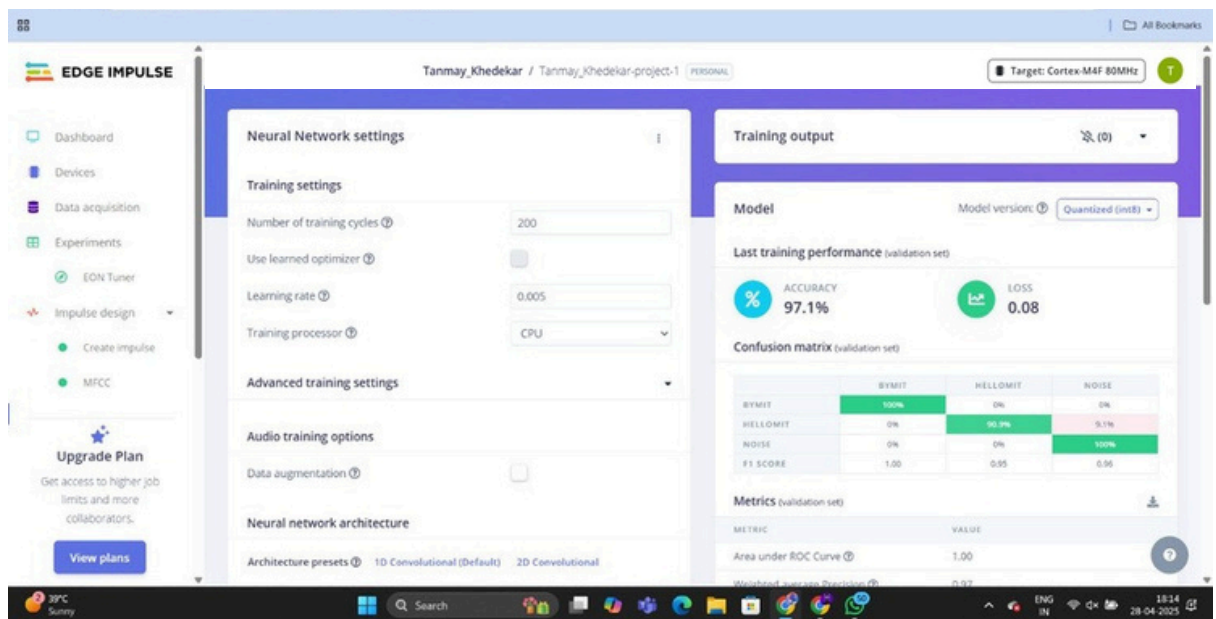
- Left Sidebar:** Contains navigation links for Dashboard, Devices, Data acquisition, Experiments, EON Tuner, and Impulse design. Under Impulse design, there are options for 'Create impulse' and 'MFCC'. An 'Upgrade Plan' button is also visible.
- Top Bar:** Shows the user's name 'Tanmay_Khedekar', the project name 'Tanmay_Khedekar-project-1', and a 'PERSONAL' label. On the right, it indicates the target device: 'Target: Cortex-M4F 80MHz'.
- Main Content Area:**
 - Dataset Overview:** Displays 'DATA COLLECTED 2m 39s' and 'TRAIN / TEST SPLIT 77% / 23%'. It includes a 'Collect data' button and a prompt to 'Connect a device to start building your dataset'.
 - Dataset Table:** A table with columns 'SAMPLE NAME', 'LABEL', 'ADDED', and 'LENGTH'. It shows a list of samples for training and testing.
 - RAW DATA:** A section with a dark background and the text 'Click on a sample to load...'.

SAMPLE NAME	LABEL	ADDED	LENGTH
noise.5nl3tkuu	noise	Apr 02 2025, 1...	5s
noise.5nl3rhc8	noise	Apr 02 2025, 1...	5s
noise.5nl3qfsf	noise	Apr 02 2025, 1...	5s
noise.5nl3p6dj	noise	Apr 02 2025, 1...	5s
hellomit.5nl219nh	hellomit	Apr 02 2025, 1...	5s
hellomit.5nl23f3o	hellomit	Apr 02 2025, 1...	5s

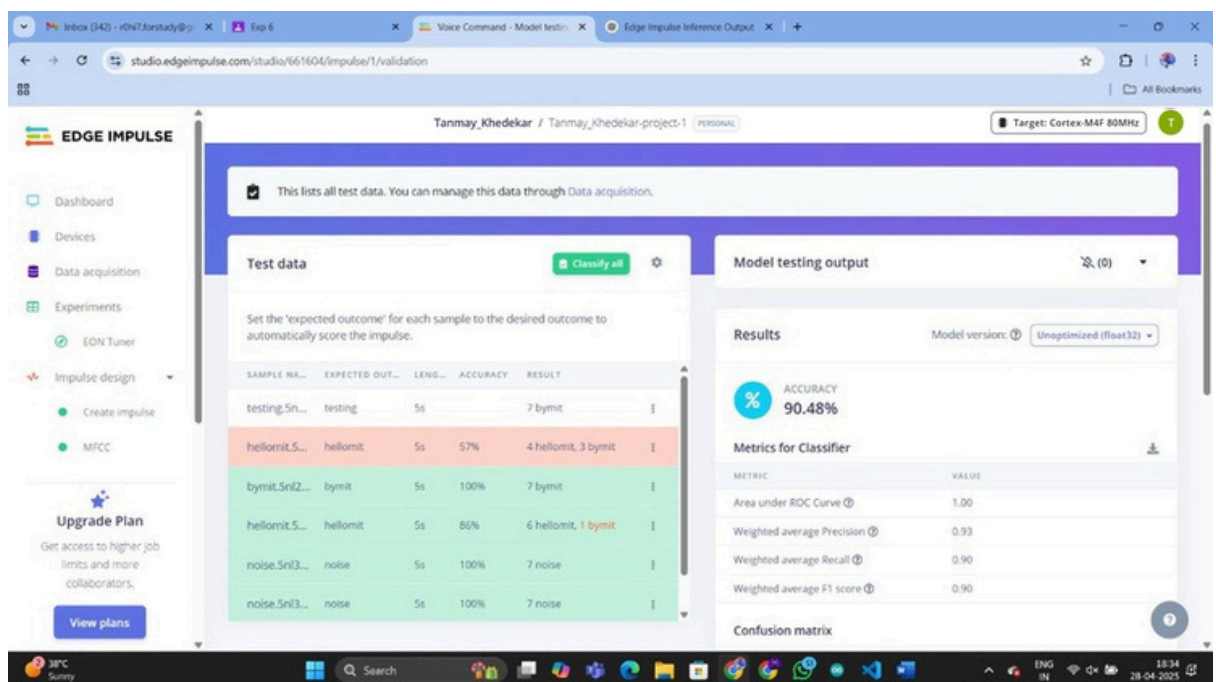
2. Feature Extraction Image



3. Accuracy / Loss Confusion Matrix Image



4. Validation Result



5. Copy of the Arduino 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.
 */

// 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 <Voice_Command_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_c
ategories[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);
        }
#ifdef 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; 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.buffer[0]);
        free(inference.buffer[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. Output

Edge Impulse Inferencing Demo

Inferencing settings:

Interval: 20.00 ms.

Frame size: 320

Sample length: 1000 ms.

No. of classes: 3

Predictions (DSP: 8 ms., Classification: 12 ms., Anomaly: 1 ms.):

hellomit: 0.85623

bymit: 0.09321

noise: 0.05056

Predictions (DSP: 7 ms., Classification: 11 ms., Anomaly: 1 ms.):

hellomit: 0.11234

bymit: 0.84219

noise: 0.04547

Predictions (DSP: 8 ms., Classification: 12 ms., Anomaly: 1 ms.):

hellomit: 0.04058

bymit: 0.02115

noise: 0.93827

Predictions (DSP: 7 ms., Classification: 12 ms., Anomaly: 1 ms.):

hellomit: 0.87129

bymit: 0.09876

noise: 0.02995

Predictions (DSP: 8 ms., Classification: 12 ms., Anomaly: 1 ms.):

hellomit: 0.05512

bymit: 0.91234

noise: 0.03254

Predictions (DSP: 7 ms., Classification: 11 ms., Anomaly: 1 ms.):

hellomit: 0.02345

bymit: 0.03487

noise: 0.94168