Edge Computing Lab

Class: TY-AIEC

School of Computing, MIT Art Design Technology University

Academic Year: 2024-25

Experiment No. 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

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:

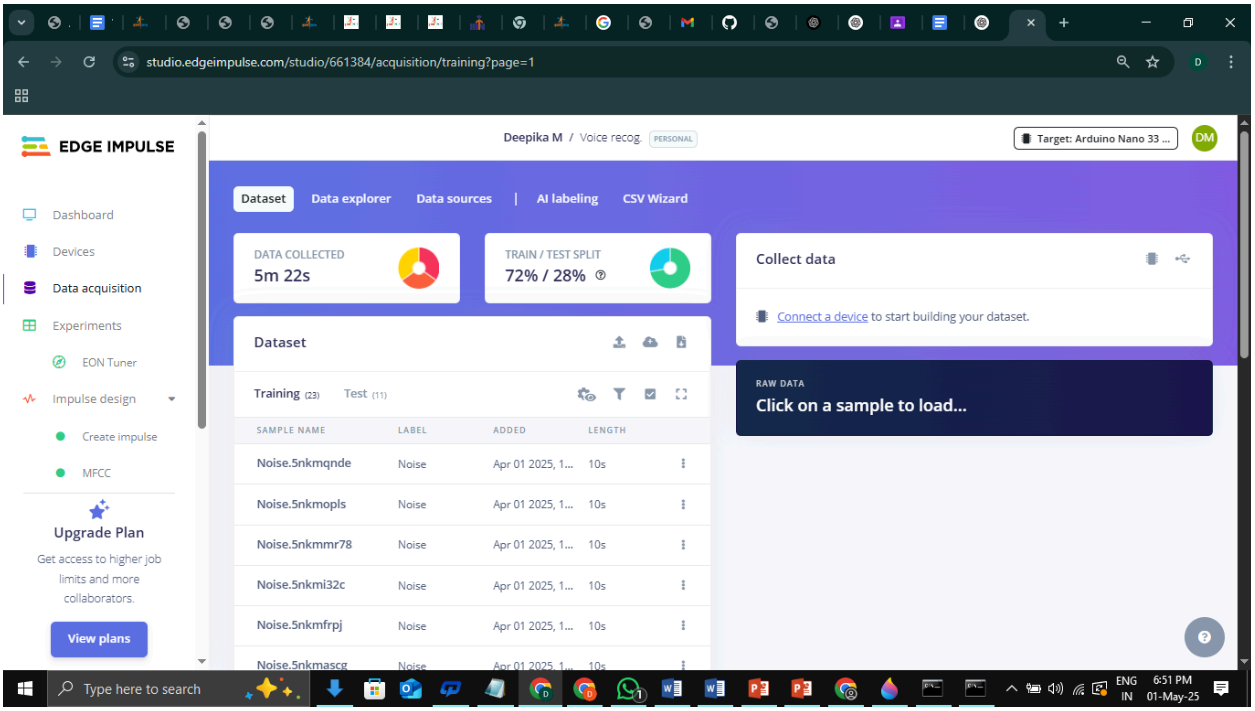
- With the model deployed, run inference on the edge device to see it classifying data in real-time.

10. Monitor:

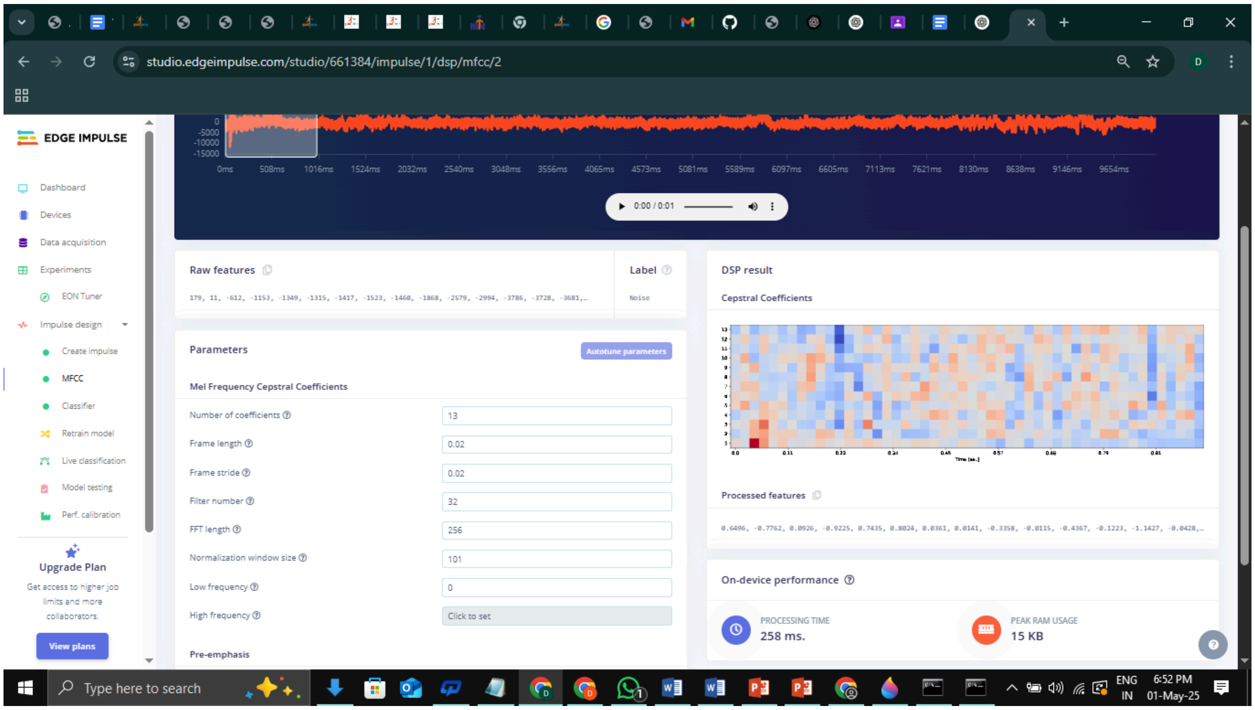
- You can monitor the performance of your device through the Edge Impulse studio.

Results:

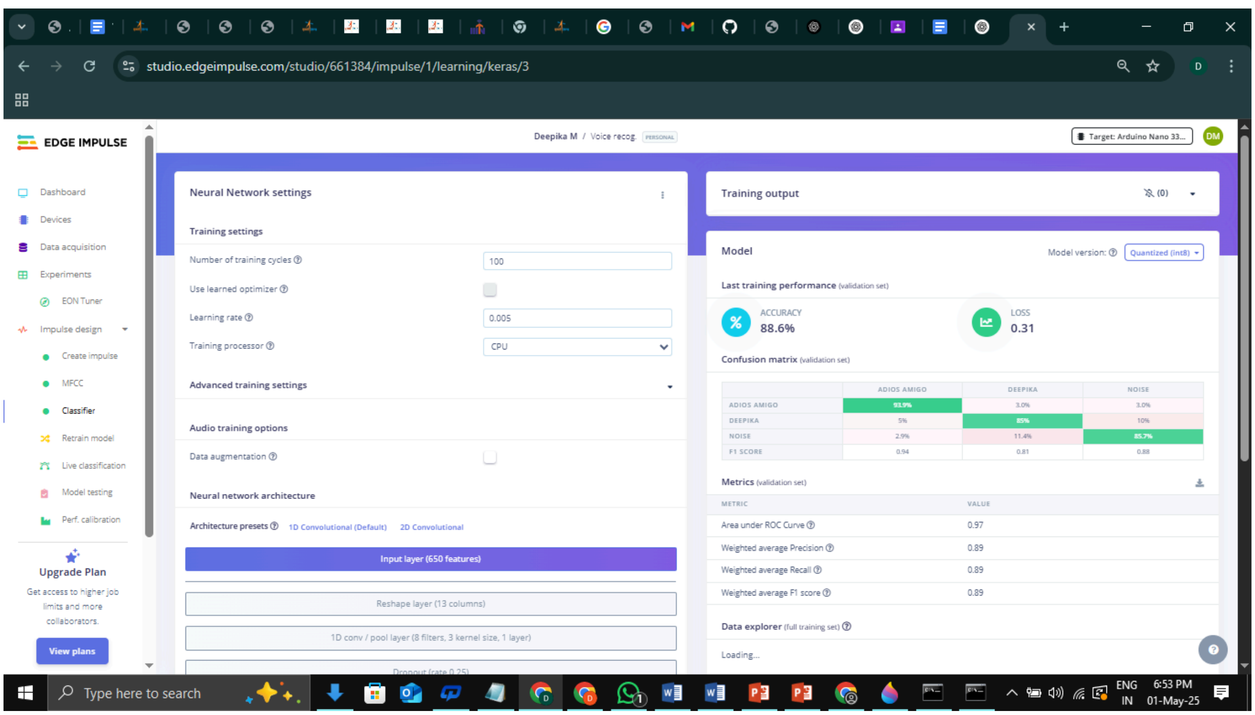
1.Dataset:

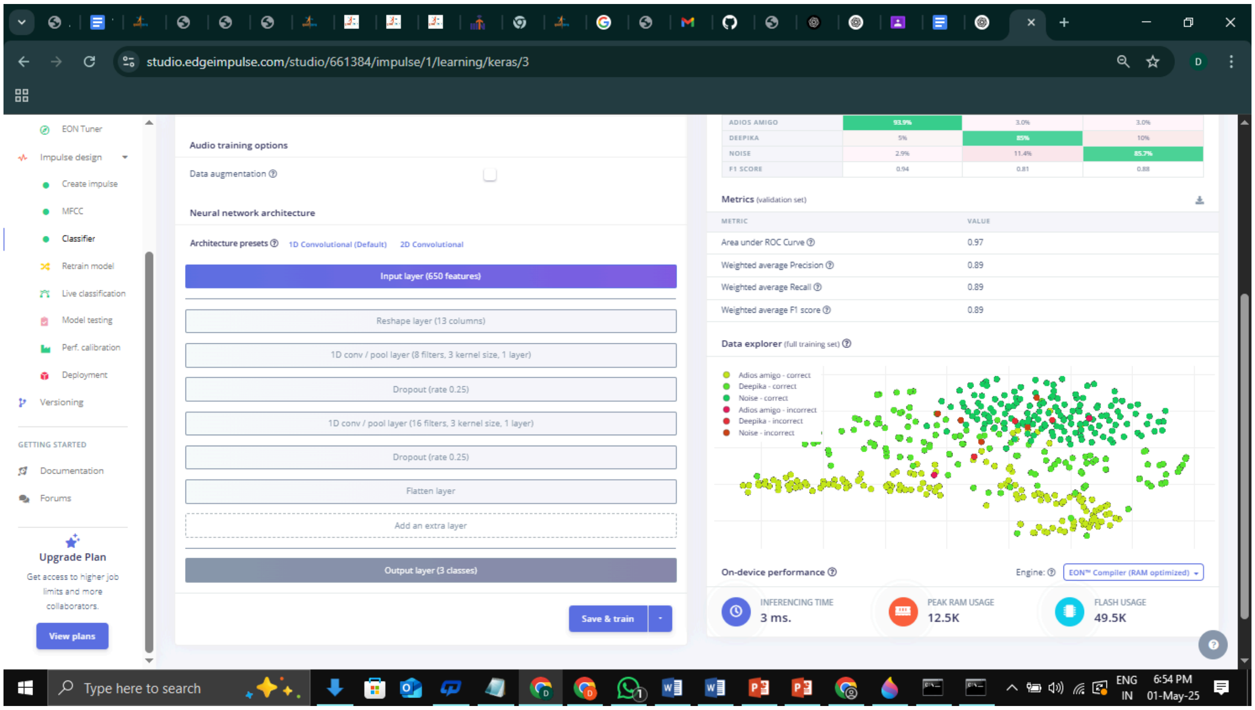


2.Feature Extraction:

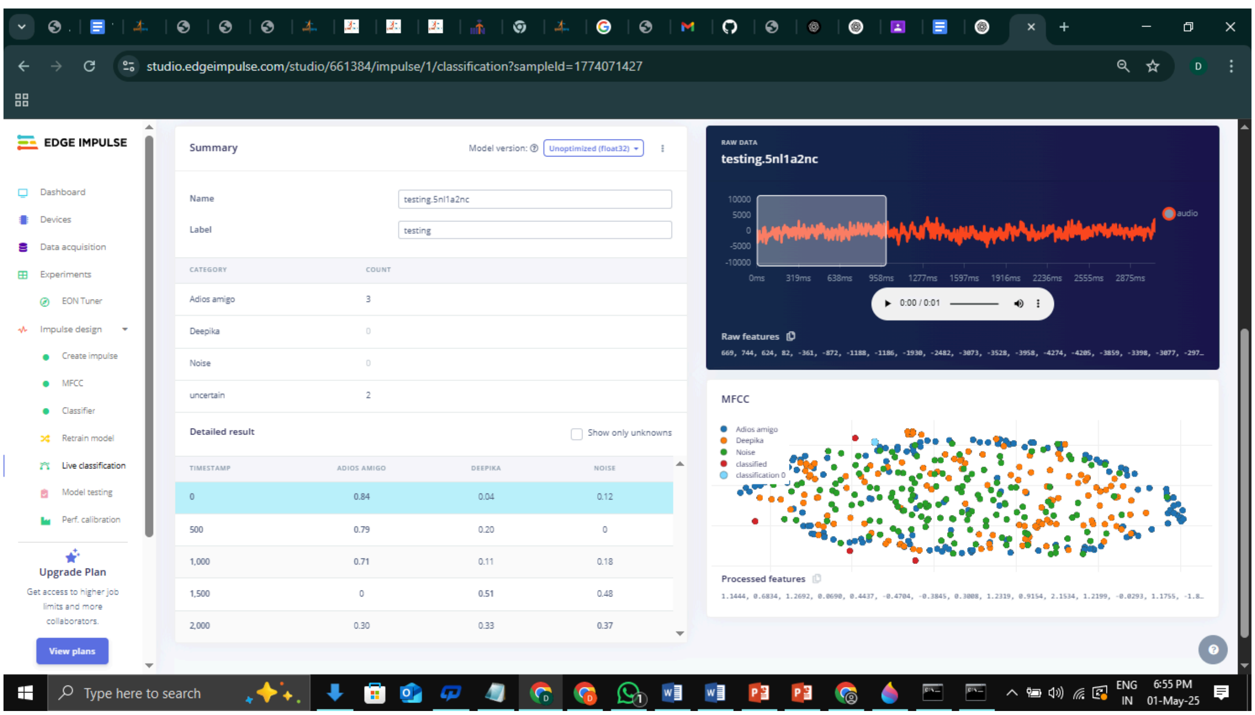


3.Accuray/Loss-Confusion Matrix:





4.Validation Result:



5.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

/*
 ** 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_recog._inferencing.h>


/** Audio buffers, pointers and selectors */
typedef struct {
    int16_t *buffer;
    uint8_t buf_ready;
    uint32_t buf_count;
    uint32_t n_samples;
} inference_t;


static inference_t inference;
static signed short sampleBuffer[2048];
static bool debug_nn = false; // Set this to true to see e.g. features
generated from the raw signal


/**
 * @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_categories[0]));


    if (microphone_inference_start(EI_CLASSIFIER_RAW_SAMPLE_COUNT) ==
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()

{

    ei_printf("Starting inferencing in 2 seconds...\n");
```

```
    delay(2000);


    ei_printf("Recording...\n");


    bool m = microphone_inference_record();
    if (!m) {
        ei_printf("ERR: Failed to record audio...\n");
        return;
    }


    ei_printf("Recording done\n");


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


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

}



/**

 * @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 (inference.buf_ready == 0) {

        for(int i = 0; i < bytesRead>>1; i++) {

            inference.buffer[inference.buf_count++] = sampleBuffer[i];



            if(inference.buf_count >= inference.n_samples) {

                inference.buf_count = 0;

                inference.buf_ready = 1;

                break;
```

```
            }
        }
    }
}

/**
 * @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.buffer = (int16_t *)malloc(n_samples * sizeof(int16_t));

    if(inference.buffer == NULL) {
        return false;
    }

    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(4096);
```

```
    // 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!");
        microphone_inference_end();

        return false;
    }

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

    return true;
}


/**
 * @brief      Wait on new data
 *
 * @return     True when finished
 */
static bool microphone_inference_record(void)
{
    inference.buf_ready = 0;
    inference.buf_count = 0;

    while(inference.buf_ready == 0) {
```

```
        delay(10);

    }


    return true;

}



/**

 * 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.buffer[offset], out_ptr, length);


    return 0;

}



/**

 * @brief      Stop PDM and release buffers

 */

static void microphone_inference_end(void)

{

    PDM.end();

    free(inference.buffer);

}



#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR !=
EI_CLASSIFIER_SENSOR_MICROPHONE

#error "Invalid model for current sensor."
```

```
#endif
```

6.Arduino terminal result:

(Forgot to take a screenshot of the running code. I’m extremely sorry.)