

IE417/EL530-Introduction to Embedded Artificial Intelligence



Lab 1

Voice Controlling LEDs with Edge Impulse

Group name : Embedded Minds

Ruturajsinh Chauhan(202101146)

Devansh Shrimali(202101492)

Raj Chauhan(202101049)

Jalp Patel(202101267)

1.Introduction

- Keyword spotting (KWS) is an essential technology used to enable hands-free interaction with devices in various daily-life applications. Popular voice assistants like "OK Google," "Alexa," "Hey Siri," and "Cortana" utilize KWS to detect specific wake-up phrases before interacting with the device.
- In this lab, we explore the application of KWS through Edge Impulse by building an application that controls a light-emitting diode (LED) based on voice commands. The commands include specifying the LED color (red, green, or blue) and the number of times it should blink (one, two, or three times). This TinyML application can be utilized in smart educational toys to teach both color and number vocabulary, offering a privacy-secure solution since it does not require internet connectivity.

2. Steps Performed

2.1 Dataset Preparation

We started by preparing the dataset, collecting audio data for the commands "blue," "red," "green," and "idle state" using an Arduino Nano BLE 33 Sense. The dataset included various utterances for each command to ensure diversity and accuracy during the training process.

2.2 Feature Extraction

The collected audio samples were processed using Mel-frequency cepstral coefficients (MFCC), a widely used feature extraction method for speech recognition. This step involved converting the raw audio data into a format that could be efficiently used by the neural network model.

2.3 Model Design and Training

We designed a neural network (NN) model and trained it using the Edge Impulse platform. During the training phase, we achieved a remarkable accuracy of 94.4%, which indicated the model's effectiveness in recognizing the specified voice commands.

2.4 Deployment

After successfully training and optimizing the model, we deployed it to the Arduino Nano BLE 33 Sense. We used the following library for deployment:

After successfully training and optimizing the model, we deployed it to the Arduino Nano BLE 33 Sense. We used the following library for deployment:

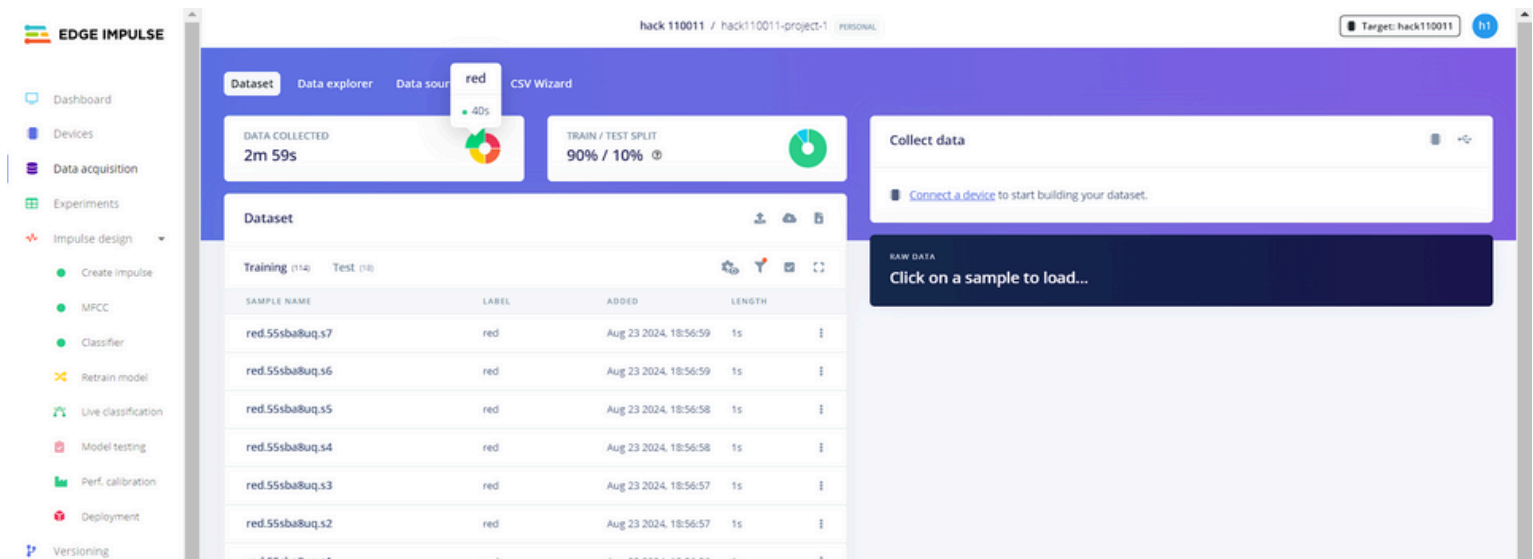
- Library Name: ei-hack110011-project1-arduino-1.0.10

We imported this library into the Arduino IDE and uploaded the code to the Arduino Nano BLE 33 Sense board using the following steps:

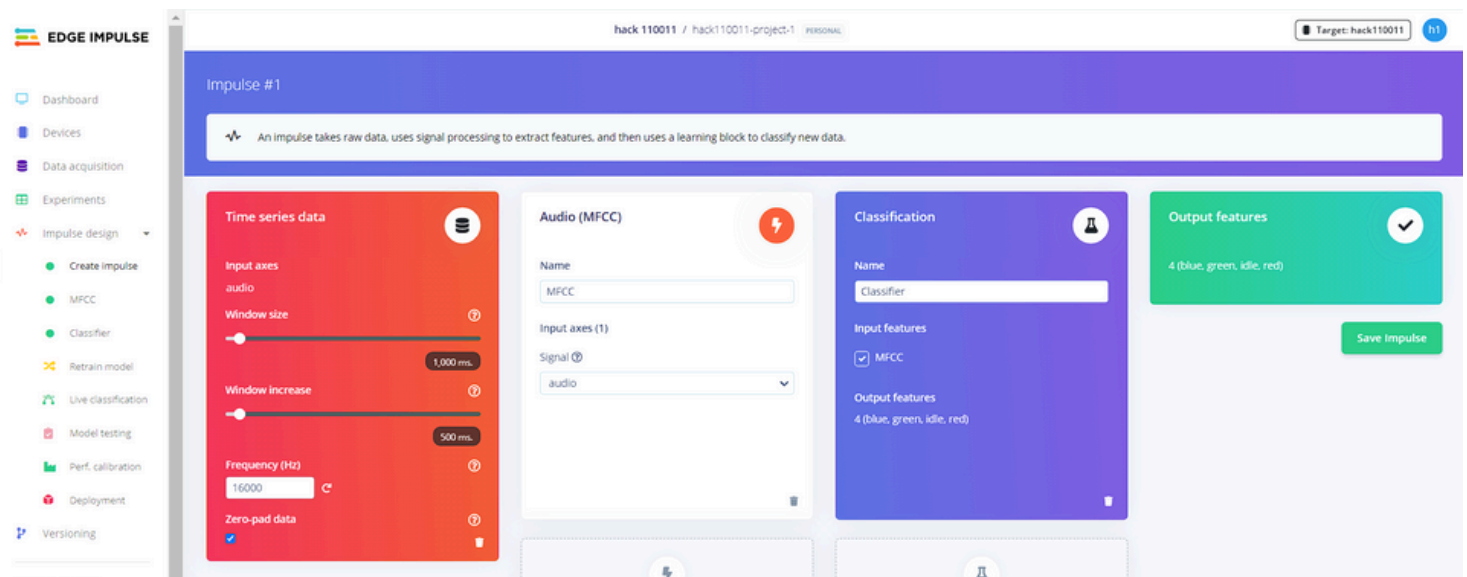
1. Arduino IDE ->File -> Examples -> hack110011-project-1_inferencing ->nano_ble33_sense -> nano_ble33_sense_microphone_continuous
- 2.Uploaded the code to the Arduino Nano BLE 33 Sense and captured the output on the serial monitor.

Output images :

Data Acquisition



Model train and classification



Dashboard

Devices

Data acquisition

Experiments

Impulse design

Create impulse

MFCC

Classifier

Retrain model

Live classification

Model testing

Perf. calibration

Deployment

Upgrade Plan

Get access to higher job limits, collaborators and a

hack110011 / hack110011-project-1

PERSONAL

Target: hack110011

Neural Network settings

Training settings

Number of training cycles

300

Use learned optimizer

Learning rate

0.005

Training processor

CPU

Advanced training settings

Validation set size

20

%

Split train/validation set on metadata key

Batch size

32

Auto-weight classes

Profile int8 model

Audio training options

Data augmentation

Training output

Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY

97.4%

LOSS

0.08

Confusion matrix (validation set)

	BLUE	GREEN	IDLE	RED
BLUE	100%	0%	0%	0%
GREEN	0%	83.3%	0%	16.7%
IDLE	0%	0%	100%	0%
RED	0%	0%	0%	100%
F1 SCORE	1.00	0.91	1.00	0.95

Metrics (validation set)

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

Neural network architecture

Architecture presets

1D Convolutional (Default)

2D Convolutional

Input layer (650 features)

Reshape layer (13 columns)

1D conv / pool layer (8 neurons, 3 kernel size, 1 layer)

Dropout (rate 0.25)

1D conv / pool layer (16 neurons, 3 kernel size, 1 layer)

Dropout (rate 0.25)

Flatten layer

Add an extra layer

Data explorer (full training set)

blue - correct

green - correct

idle - correct

red - correct

green - incorrect

On-device performance

Engine: TensorFlow Lite

INFERRING TIME

4 ms.

PEAK RAM USAGE

6.4K

FLASH USAGE

52.2K

Model testing

Dashboard

Devices

Data acquisition

Experiments

Impulse design

Create impulse

MFCC

Classifier

Retrain model

Live classification

Model testing

Perf. calibration

Deployment

Upgrade Plan

Get access to higher job limits, collaborators and a full commercial license.

hack110011 / hack110011-project-1

PERSONAL

Target: hack110011

Test data

Classify all

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

SAMPLE NAME	EXPECTED OUTCOME	LENGTH	ACCURACY	RESULT
red.55sg5m0h...	red	1s	100%	1 red
red.55sg5m0h...	red	1s	0%	1 blue
red.55sf71n7.s3	red	1s	100%	1 red
red.55sf71n7.s1	red	1s	100%	1 red
idle.55se0vsl.s3	idle	1s	100%	1 idle
idle.55se0vsl.s2	idle	1s	100%	1 idle
idle.55se0vsl.s1	idle	1s	100%	1 idle
green.55sdpsg...	green	1s	100%	1 green
green.55sdpsg...	green	1s	100%	1 green
green.55sdpsg...	green	1s	100%	1 green
green.55sdpsg...	green	1s	100%	1 green
green.55sdpsg...	green	1s	100%	1 green
green.55sdpsg...	green	1s	100%	1 green
blue.55sdlk5.s6	blue	1s	100%	1 blue

Model testing output

Model version: Unoptimized (float32)

Results

ACCURACY

94.44%

Metrics for Classifier

METRIC	VALUE
Area under ROC Curve	0.98
Weighted average Precision	0.95
Weighted average Recall	0.94
Weighted average F1 score	0.94

Confusion matrix

	BLUE	GREEN	IDLE	RED	UNCERTAIN
BLUE	100%	0%	0%	0%	0%
GREEN	0%	100%	0%	0%	0%
IDLE	0%	0%	100%	0%	0%
RED	25%	0%	0%	75%	0%
F1 SCORE	0.92	1.00	1.00	0.86	

Feature explorer

blue - correct

green - correct

idle - correct

red - correct

red - incorrect

3. Output and Results

The final trained model achieved a 94.4% accuracy during testing. The deployment on the Arduino Nano BLE 33 Sense allowed us to control the LED based on voice commands, demonstrating the practical application of the developed KWS system.

4. Conclusion

This lab successfully demonstrated the development of an end-to-end KWS application using Edge Impulse. The project provided hands-on experience with audio data acquisition, feature extraction using MFCC, neural network training, model optimization, and deployment on low-power hardware like the Arduino Nano BLE 33 Sense. The achieved accuracy of 94.4% showcases the efficiency of the developed model in real-time voice-controlled applications.

Video Link

Serial monitor output:

<https://drive.google.com/file/d/1bGjv4Z18J6YCAXL-i-Lzxc3rfyQZaonA/view>

Demo Video:

<https://drive.google.com/file/d/1Og17w0UvQneIYDnbpW71T3LwHNhTiE-K/view?usp=sharing>

code:

```
#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 3

/* Includes ----- */
#include <PDM.h>
#include <hack110011-project-1_inferencing.h>

#define RED 22
#define BLUE 24
#define GREEN 23
#define LED_PWR 25

/** 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()
{
    pinMode(RED, OUTPUT);
    pinMode(BLUE, OUTPUT);
    pinMode(GREEN, OUTPUT);
}
```



```

pinMode(LED_PWR, OUTPUT);
//pinMode(LED_PWR, OUTPUT);

// put your setup code here, to run once:
Serial.begin(115200);

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]));

run_classifier_init();
if (microphone_inference_start(EI_CLASSIFIER_SLICE_SIZE) == false) {
    ei_printf("ERR: Failed to setup audio sampling\r\n");
    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};

```

```

79
80 EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, debug_nn);
81 if (r != EI_IMPULSE_OK) {
82     ei_printf("ERR: Failed to run classifier (%d)\n", r);
83     return;
84 }
85
86 if (result.classification[0].value >= 0.7) {
87     digitalWrite(BLUE, LOW);
88     digitalWrite(GREEN, HIGH);
89     digitalWrite(RED, HIGH);
90 }
91 else if (result.classification[1].value >= 0.7) {
92     digitalWrite(BLUE, HIGH);
93     digitalWrite(GREEN, LOW);
94     digitalWrite(RED, HIGH);
95 }
96 else if (result.classification[2].value >= 0.7) {
97     digitalWrite(BLUE, HIGH);
98     digitalWrite(GREEN, HIGH);
99     digitalWrite(RED, HIGH);
100 }
101 else if (result.classification[3].value >= 0.7) {
102     digitalWrite(BLUE, HIGH);
103     digitalWrite(GREEN, HIGH);
104     digitalWrite(RED, LOW);
105 }
106
107 if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)) {
108     // print the predictions
109     ei_printf("Predictions ");
110     ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
111         result.timing.dsp, result.timing.classification, result.timing.anomaly);
112     ei_printf(": \n");
113     for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
114         ei_printf("    %s: %.5f\n", result.classification[ix].label,
115             result.classification[ix].value);
116     }
117 #if EI_CLASSIFIER_HAS_ANOMALY == 1
118     ei_printf("    anomaly score: %.3f\n", result.anomaly);
119 #endif
120

```

```

121 |     print_results = 0;
122 | }
123 | }
124 |
125 | /**
126 |  * @brief      Printf function uses vsnprintf and output using Arduino Serial
127 |  * @param[in]  format      Variable argument list
128 |  */
129 | void ei_printf(const char *format, ...) {
130 |     static char print_buf[1024] = { 0 };
131 |
132 |     va_list args;
133 |     va_start(args, format);
134 |     int r = vsnprintf(print_buf, sizeof(print_buf), format, args);
135 |     va_end(args);
136 |
137 |     if (r > 0) {
138 |         Serial.write(print_buf);
139 |     }
140 | }
141 |
142 | /**
143 |  * @brief      PDM buffer full callback
144 |  * | | | | | | | | | | Get data and call audio thread callback
145 |  */
146 | static void pdm_data_ready_inference_callback(void)
147 | {
148 |     int bytesAvailable = PDM.available();
149 |
150 |     // read into the sample buffer
151 |     int bytesRead = PDM.read((char *)&sampleBuffer[0], bytesAvailable);
152 |
153 |     if (record_ready == true) {
154 |         for (int i = 0; i < bytesRead >> 1; i++) {
155 |             inference.buffers[inference.buf_select][inference.buf_count++] = sampleBuffer[i];
156 |
157 |             if (inference.buf_count >= inference.n_samples) {
158 |                 inference.buf_select ^= 1;
159 |                 inference.buf_count = 0;
160 |                 inference.buf_ready = 1;
161 |             }

```

```

161     }
162 }
163 }
164 }
165
166 /**
167  @brief      Init inferencing struct and setup/start PDM
168  @param[in]  n_samples  The n samples
169  @return     { description_of_the_return_value }
170 */
171 static bool microphone_inference_start(uint32_t n_samples)
172 {
173     inference.buffers[0] = (signed short *)malloc(n_samples * sizeof(signed short));
174
175     if (inference.buffers[0] == NULL) {
176         return false;
177     }
178
179     inference.buffers[1] = (signed short *)malloc(n_samples * sizeof(signed short));
180
181     if (inference.buffers[0] == NULL) {
182         free(inference.buffers[0]);
183         return false;
184     }
185
186     sampleBuffer = (signed short *)malloc((n_samples >> 1) * sizeof(signed short));
187
188     if (sampleBuffer == NULL) {
189         free(inference.buffers[0]);
190         free(inference.buffers[1]);
191         return false;
192     }
193
194     inference.buf_select = 0;
195     inference.buf_count = 0;
196     inference.n_samples = n_samples;
197     inference.buf_ready = 0;
198
199     // configure the data receive callback
200     PDM.onReceive(&pdm_data_ready_inference_callback);
201

```

```

201
202     PDM.setBufferSize((n_samples >> 1) * sizeof(int16_t));
203
204     // initialize PDM with:
205     // - one channel (mono mode)
206     // - a 16 kHz sample rate
207     if (!PDM.begin(1, EI_CLASSIFIER_FREQUENCY)) {
208         ei_printf("Failed to start PDM!");
209     }
210
211     // set the gain, defaults to 20
212     PDM.setGain(127);
213
214     record_ready = true;
215
216     return true;
217 }
218
219 /**
220  * @brief      Wait on new data
221  * @return     True when finished
222  */
223 static bool microphone_inference_record(void)
224 {
225     bool ret = true;
226
227     if (inference.buf_ready == 1) {
228         ei_printf(
229             "Error sample buffer overrun. Decrease the number of slices per model window "
230             "(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)\n");
231         ret = false;
232     }
233
234     while (inference.buf_ready == 0) {
235         delay(1);
236     }
237
238     inference.buf_ready = 0;
239
240     return ret;
241 }

```

```
242
243 /**
244  | Get raw audio signal data
245 */
246 static int microphone_audio_signal_get_data(size_t offset, size_t length, float *out_ptr)
247 {
248     numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1][offset], out_ptr, length);
249
250     return 0;
251 }
252
253 /**
254  | @brief Stop PDM and release buffers
255 */
256 static void microphone_inference_end(void)
257 {
258     PDM.end();
259     free(inference.buffers[0]);
260     free(inference.buffers[1]);
261     free(sampleBuffer);
262 }
263
264 #if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR != EI_CLASSIFIER_SENSOR_MICROPHONE
265 #error "Invalid model for current sensor."
266 #endif
```