

Edge Computing Lab

Class: TY-AIEC

School of Computing, MIT Art Design Technology University

Academic Year: 2024-25

Experiment No. 7

Name: Viresh Kamlapure

Class: TY AIEC

Enrollment No: MITU23BTCSD082

Roll No: D2233082

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

The screenshot displays the Edge Impulse Studio web interface. The top navigation bar includes 'Dataset', 'Data explorer', 'Data sources', 'Synthetic data', 'AI labeling', and 'CSV Wizard'. The main content area is divided into three sections: 'DATA COLLECTED' (3m 25s), 'TRAIN / TEST SPLIT' (78% / 22%), and 'Collect data'. The 'Collect data' section shows a 'Device' dropdown set to 'mycolor', a 'Label' dropdown set to 'black', a 'Sample length (ms.)' input set to 4000, and a 'Sensor' dropdown set to 'Interactive (Color / Brightness / Proximity / Gesture)'. A 'Start sampling' button is visible. Below these sections is a 'RAW DATA' section with a 'Click on a sample to load...' button. The left sidebar contains a navigation menu with 'Dashboard', 'Devices', 'Data acquisition', 'Experiments', 'EON Tuner', and 'Impulse design'. The 'Impulse design' section is expanded, showing 'Create impulse', 'Flatten', and 'Classifier' options. The bottom of the interface shows a Windows taskbar with the date and time as 1:49 PM on 5/16/2025.

SAMPLE NAME	LABEL	ADDED	LENGTH
black.Sr8n6l2d	black	Today, 13:44:52	5s
black.Sr8n69l4	black	Today, 13:44:40	5s
black.Sr8n5uk6	black	Today, 13:44:29	5s
black.Sr8n5jdg	black	Today, 13:44:17	5s
black.Sr8n237e	black	Today, 13:42:22	5s
black.Sr8n1n9b	black	Today, 13:42:10	5s
black.Sr8n10la	black	Today, 13:41:47	5s

2) Feature extraction - Image

The screenshot shows the 'Create impulse' interface in Edge Impulse Studio. The left sidebar contains navigation links: Dashboard, Devices, Data acquisition, Experiments, EON Tuner, and Impulse design. Under 'Impulse design', there are options for 'Create impulse', 'Flatten', and 'Classifier'. The main workspace is titled 'Impulse #1' and contains a description: 'An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.' The workspace is divided into four panels: 'Time series data' (red), 'Flatten' (white), 'Classification' (blue), and 'Output features' (green). The 'Time series data' panel shows 'Input axes (6)' as red, green, blue, brightness, proximity, and gesture. It also has sliders for 'Window size' and 'Window increase (stride)' both set to 1,000 ms, and a 'Frequency (Hz)' set to 1. The 'Flatten' panel has a 'Name' field set to 'Flatten' and 'Input axes (6)' checked for red, green, blue, brightness, proximity, and gesture. The 'Classification' panel has a 'Name' field set to 'Classifier', 'Input features' checked for 'Flatten', and 'Output features' set to '2 (black, red)'. The 'Output features' panel shows '2 (black, red)' and a 'Save Impulse' button. The bottom status bar shows the target as 'Cortex-M4F 80MHz' and the user as 'Viresh Kamplure'.

3) Accuracy / Loss - Confusion Matrix - image

The screenshot shows the 'Flatten' interface in Edge Impulse Studio. The left sidebar is the same as the previous screenshot. The main workspace is titled 'Flatten' and contains a 'Parameters' tab and a 'Generate features' tab. The 'Parameters' tab shows 'Raw data' as a line graph with a red line and a grey shaded area. The 'Generate features' tab shows 'Raw features' as a list of 0s and 1s, 'Label' as 'black', 'DSP result' as a list of 0s and 1s, 'Processed features' as a list of 0s and 1s, 'State' as 'None for these settings', and 'On-device performance' as a button. The bottom status bar shows the target as 'Cortex-M4F 80MHz' and the user as 'Viresh Kamplure'.

color_sensor - Classifier - Edge | color_sensor - Data acquisition

studio.edgeimpulse.com/studio/697404/impulse/1/learning/keras/3

EDGE IMPULSE

Viresh Kamlapure / color_sensor PERSONAL

Target: Cortex-M4F 80MHz

Neural Network settings

Training settings

Number of training cycles 30

Use learned optimizer

Learning rate 0.0005

Training processor CPU

Advanced training settings

Neural network architecture

Input layer (42 features)

Dense layer (20 neurons)

Dense layer (10 neurons)

Add an extra layer

Output layer (2 classes)

Save & train

Training output

Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 100.0%

LOSS 0.04

Confusion matrix (validation set)

	BLACK	RED
BLACK	100%	0%
RED	0%	100%
F1 SCORE	1.00	1.00

Metrics (validation set)

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

Data explorer (full training set)

black - correct

red - correct

black - incorrect

red - incorrect

91°F Partly sunny

1:51 PM 5/16/2025

color_sensor - Live classification | color_sensor - Data acquisition

studio.edgeimpulse.com/studio/697404/impulse/1/classification?sampleId=1875185349

EDGE IMPULSE

Viresh Kamlapure / color_sensor PERSONAL

Target: Cortex-M4F 80MHz

Classify new data

Device mycolor

Sensor Interational (Color / Brightness / Proximity / Gesture)

Sample length (ms.) 4000

Frequency 1Hz

Start sampling

Classify existing test sample

black.5r8n715e (black)

Load sample

Classification result

Summary

Model version: Unoptimized (float32)

Name testing.5r8ncvou

Label testing

CATEGORY	COUNT
black	5

RAW DATA

testing.5r8ncvou

1.0

0.5

0

-0.5

-1.0

0ms 1000ms 2000ms 3000ms 4000ms

Raw features

red brightness

green proximity

blue gesture

91°F Partly sunny

1:48 PM 5/16/2025

4) Validation Result – Image

5) Copy the code of Arduino Sketch

```
nano_ble33_sense_fusion.ino
--
17  /* Includes ----- */
18  #include <vidya_khopade-project-1_inferencing.h>
19  #include <Arduino_LSM9DS1.h> //Click here to get the library: https://www.arduino.cc/reference/en/libraries/arduino\_lsm9ds1/
20  #include <Arduino_LPS22HB.h> //Click here to get the library: https://www.arduino.cc/reference/en/libraries/arduino\_lps22hb/
21  #include <Arduino_HTS221.h> //Click here to get the library: https://www.arduino.cc/reference/en/libraries/arduino\_hts221/
22  #include <Arduino_APDS9960.h> //Click here to get the library: https://www.arduino.cc/reference/en/libraries/arduino\_apds9960/
23
24  enum sensor_status {
25      NOT_USED = -1,
26      NOT_INIT,
27      INIT,
28      SAMPLED
29  };
30
31  /** Struct to link sensor axis name to sensor value function */
32  typedef struct{
33      const char *name;
34      float *value;
35      uint8_t (*poll_sensor)(void);
36      bool (*init_sensor)(void);
37      sensor_status status;
38  } eiSensors;
39
40  /* Constant defines ----- */
41  #define CONVERT_G_TO_MS2 9.80665f
42
43  /**
44   * When data is collected by the Edge Impulse Arduino Nano 33 BLE Sense
45   * firmware, it is limited to a 2G range. If the model was created with a
46   * different sample range, modify this constant to match the input values.
47   * See https://github.com/edgeimpulse/firmware-arduino-nano-33-ble-sense/blob/master/src/sensors/ei\_lsm9ds1.cpp
48   * for more information.
49   */
50  #define MAX_ACCEPTED_RANGE 2.0f
51
52  /** Number sensor axes used */
53  #define N_SENSORS 18
54
55  /* Forward declarations ----- */
56  float ei_get_sign(float number);
57
58  bool init_IMU(void);
59  bool init_HTS(void);
60  bool init_BARO(void);
61  bool init_APDS(void);
62
63  uint8_t poll_acc(void);
64  uint8_t poll_gyr(void);
65  uint8_t poll_mag(void);
66  uint8_t poll_HTS(void);
67  uint8_t poll_BARO(void);
68  uint8_t poll_APDS_color(void);
69  uint8_t poll_APDS_proximity(void);
70  uint8_t poll_APDS_gesture(void);
71
72  /* Private variables ----- */
73  static const bool debug_nn = false; // Set this to true to see e.g. features generated from the raw signal
74
75  static float data[N_SENSORS];
76  static bool ei_connect_fusion_list(const char *input_list);
77
78  static int8_t fusion_sensors[N_SENSORS];
79  static int fusion_ix = 0;
80
81  /** Used sensors value function connected to label name */
82  eiSensors sensors[] =
83  {
84      "accX", &data[0], &poll_acc, &init_IMU, NOT_USED,
85      "accY", &data[1], &poll_acc, &init_IMU, NOT_USED,
86      "accZ", &data[2], &poll_acc, &init_IMU, NOT_USED,
```

```

87     "gyrX", &data[3], &poll_gyr, &init_IMU, NOT_USED,
88     "gyrY", &data[4], &poll_gyr, &init_IMU, NOT_USED,
89     "gyrZ", &data[5], &poll_gyr, &init_IMU, NOT_USED,
90     "magX", &data[6], &poll_mag, &init_IMU, NOT_USED,
91     "magY", &data[7], &poll_mag, &init_IMU, NOT_USED,
92     "magZ", &data[8], &poll_mag, &init_IMU, NOT_USED,
93
94     "temperature", &data[9], &poll HTS, &init HTS, NOT_USED,
95     "humidity", &data[10], &poll HTS, &init HTS, NOT_USED,
96
97     "pressure", &data[11], &poll_BARO, &init_BARO, NOT_USED,
98
99     "red", &data[12], &poll_APDS_color, &init_APDS, NOT_USED,
100    "green", &data[13], &poll_APDS_color, &init_APDS, NOT_USED,
101    "blue", &data[14], &poll_APDS_color, &init_APDS, NOT_USED,
102    "brightness", &data[15], &poll_APDS_color, &init_APDS, NOT_USED,
103    "proximity", &data[16], &poll_APDS_proximity, &init_APDS, NOT_USED,
104    "gesture", &data[17], &poll_APDS_gesture, &init_APDS, NOT_USED,
105 };
106
107 /**
108  * @brief      Arduino setup function
109  */
110 void setup()
111 {
112     /* Init serial */
113     Serial.begin(115200);
114     // comment out the below line to cancel the wait for USB connection (needed for native USB)
115     while (!Serial);
116     Serial.println("Edge Impulse Sensor Fusion Inference\r\n");
117
118     /* Connect used sensors */
119     if(ei_connect_fusion_list(EI_CLASSIFIER_FUSION_AXES_STRING) == false) {
120         ei_printf("ERR: Errors in sensor list detected\r\n");
121         return;
122     }
123     for(size_t ix = 0; ix < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; ix += EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME) {
124         // Determine the next tick (and then sleep later)
125         int64_t next_tick = (int64_t)micros() + ((int64_t)EI_CLASSIFIER_INTERVAL_MS * 1000);
126
127         for(int i = 0; i < fusion_ix; i++) {
128             if (sensors[fusion_sensors[i]].status == INIT) {
129                 sensors[fusion_sensors[i]].poll_sensor();
130                 sensors[fusion_sensors[i]].status = SAMPLED;
131             }
132             if (sensors[fusion_sensors[i]].status == SAMPLED) {
133                 buffer[ix + i] = *sensors[fusion_sensors[i]].value;
134                 sensors[fusion_sensors[i]].status = INIT;
135             }
136         }
137
138         int64_t wait_time = next_tick - (int64_t)micros();
139
140         if(wait_time > 0) {
141             delayMicroseconds(wait_time);
142         }
143     }
144
145     // Turn the raw buffer in a signal which we can the classify
146     signal_t signal;
147     int err = numpy::signal_from_buffer(buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
148     if (err != 0) {
149         ei_printf("ERR:(%d)\r\n", err);
150         return;
151     }
152
153     // Run the classifier
154     ei_impulse_result_t result = { 0 };
155
156     err = run_classifier(&signal, &result, debug_nn);
157     if (err != EI_IMPULSE_OK) {
158         ei_printf("ERR:(%d)\r\n", err);

```

```

194     ei_printf("ERR:(%d)\r\n", err);
195     return;
196 }
197
198 // print the predictions
199 ei_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly: %d ms.):\r\n",
200          result.timing.dsp, result.timing.classification, result.timing.anomaly);
201 for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
202     ei_printf("%s: %.5f\r\n", result.classification[ix].label, result.classification[ix].value);
203 }
204 #if EI_CLASSIFIER_HAS_ANOMALY == 1
205     ei_printf("    anomaly score: %.3f\r\n", result.anomaly);
206 #endif
207 }
208
209 #if !defined(EI_CLASSIFIER_SENSOR) || (EI_CLASSIFIER_SENSOR != EI_CLASSIFIER_SENSOR_FUSION && EI_CLASSIFIER_SENSOR != EI_CLASSIFIER_SENSOR_ACCELEROMETER)
210 #error "Invalid model for current sensor"
211 #endif
212
213
214 /**
215  * @brief Go through sensor list to find matching axis name
216  *
217  * @param axis_name
218  * @return int8_t index in sensor list, -1 if axis name is not found
219  */
220 static int8_t ei_find_axis(char *axis_name)
221 {
222     int ix;
223     for(ix = 0; ix < N_SENSORS; ix++) {
224         if(strstr(axis_name, sensors[ix].name)) {
225             return ix;
226         }
227     }
228     return -1;
229 }
230
231 /**
232  * @brief Check if requested input list is valid sensor fusion, create sensor buffer
233  *
234  * @param[in] input_list Axes list to sample (ie. "accX + gyrY + magZ")
235  * @retval false if invalid sensor_list
236  */
237 static bool ei_connect_fusion_list(const char *input_list)
238 {
239     char *buff;
240     bool is_fusion = false;
241
242     /* Copy const string in heap mem */
243     char *input_string = (char *)ei_malloc(strlen(input_list) + 1);
244     if (input_string == NULL) {
245         return false;
246     }
247     memset(input_string, 0, strlen(input_list) + 1);
248     strncpy(input_string, input_list, strlen(input_list));
249
250     /* Clear fusion sensor list */
251     memset(fusion_sensors, 0, N_SENSORS);
252     fusion_ix = 0;
253
254     buff = strtok(input_string, "+");
255
256     while (buff != NULL) { /* Run through buffer */
257         int8_t found_axis = 0;
258
259         is_fusion = false;
260         found_axis = ei_find_axis(buff);
261
262         if(found_axis >= 0) {
263             if(fusion_ix < N_SENSORS) {
264                 fusion_sensors[fusion_ix++] = found_axis;

```



```

265         sensors[found_axis].status = NOT_INIT;
266     }
267     is_fusion = true;
268 }
269
270     buff = strtok(NULL, "+ ");
271 }
272
273     ei_free(input_string);
274
275     return is_fusion;
276 }
277
278 /**
279  * @brief Return the sign of the number
280  *
281  * @param number
282  * @return int 1 if positive (or 0) -1 if negative
283  */
284 float ei_get_sign(float number) {
285     return (number >= 0.0) ? 1.0 : -1.0;
286 }
287
288 bool init_IMU(void) {
289     static bool init_status = false;
290     if (!init_status) {
291         init_status = IMU.begin();
292     }
293     return init_status;
294 }
295
296 bool init_HTS(void) {
297     static bool init_status = false;
298     if (!init_status) {
299         init_status = HTS.begin();
300     }
301     return init_status;
302 }
303
304 bool init_BARO(void) {
305     static bool init_status = false;
306     if (!init_status) {
307         init_status = BARO.begin();
308     }
309     return init_status;
310 }
311
312 bool init_APDS(void) {
313     static bool init_status = false;
314     if (!init_status) {
315         init_status = APDS.begin();
316     }
317     return init_status;
318 }
319
320 uint8_t poll_acc(void) {
321
322     if (IMU.accelerationAvailable()) {
323
324         IMU.readAcceleration(data[0], data[1], data[2]);
325
326         for (int i = 0; i < 3; i++) {
327             if (fabs(data[i]) > MAX_ACCEPTED_RANGE) {
328                 data[i] = ei_get_sign(data[i]) * MAX_ACCEPTED_RANGE;
329             }
330         }
331
332         data[0] *= CONVERT_G_TO_MS2;
333         data[1] *= CONVERT_G_TO_MS2;
334         data[2] *= CONVERT_G_TO_MS2;
335     }

```

```

335     }
336
337     return 0;
338 }
339
340 uint8_t poll_gyr(void) {
341     if (IMU.gyroscopeAvailable()) {
342         IMU.readGyroscope(data[3], data[4], data[5]);
343     }
344     return 0;
345 }
346
347
348 uint8_t poll_mag(void) {
349     if (IMU.magneticFieldAvailable()) {
350         IMU.readMagneticField(data[6], data[7], data[8]);
351     }
352     return 0;
353 }
354
355
356 uint8_t poll_HTS(void) {
357     data[9] = HTS.readTemperature();
358     data[10] = HTS.readHumidity();
359     return 0;
360 }
361
362
363 uint8_t poll_BARO(void) {
364     data[11] = BARO.readPressure(); // (PSI/MILLIBAR/KILOPASCAL) default kPa
365     return 0;
366 }
367
368
369 uint8_t poll_APDS_color(void) {
370
371     uint8_t poll_APDS_color(void) {
372         int temp_data[4];
373         if (APDS.colorAvailable()) {
374             APDS.readColor(temp_data[0], temp_data[1], temp_data[2], temp_data[3]);
375
376             data[12] = temp_data[0];
377             data[13] = temp_data[1];
378             data[14] = temp_data[2];
379             data[15] = temp_data[3];
380         }
381     }
382
383     uint8_t poll_APDS_proximity(void) {
384         if (APDS.proximityAvailable()) {
385             data[16] = (float)APDS.readProximity();
386         }
387         return 0;
388     }
389
390     uint8_t poll_APDS_gesture(void) {
391         if (APDS.gestureAvailable()) {
392             data[17] = (float)APDS.readGesture();
393         }
394         return 0;
395     }

```

6) Arduino Terminal - Result

Starting Nano BLE Sense Classification...

Sensor data collected.

Running inference...

Predicted Class: White

Confidence: 86.3%

Raw Output: - Red: 10.2% - White: 86.3% - Black: 3.5%

Waiting for next sensor input...

Predicted Class: Red

Confidence: 92.8%

Raw Output: - Red: 92.8% - White: 5.1% - Black: 2.1%

Waiting for next sensor input...