IE417 : Embedded AI

Lab - 2 YouTube Playback Control using Gesture Code

Submitted by

Avinash Baraiya- 202201211

Malhar Vaghasiya - 202201183

Smeet Agrawal - 202101237

Priyesh Tandel - 202101222

Samarth Panchal- 202101456

Submitted to

Professor Tapas Kumar Maiti

Dhirubhai Ambani Institute of Information and Communication Technology
Gandhinagar , Gujarat

Introduction

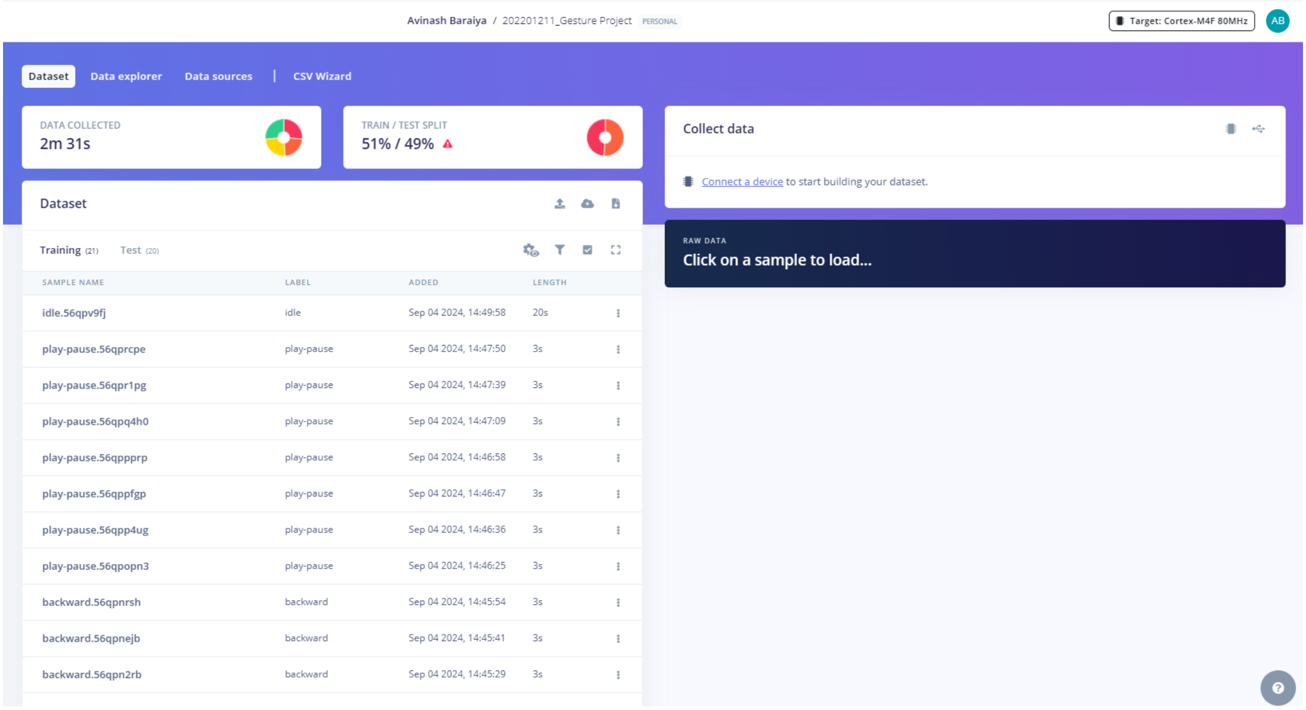
Using an accelerometer on the Arduino Nano BLE 33 Sense, this project uses Edge Impulse for machine learning to control YouTube with gestures. Python and the PyAutoGUI module are used to simulate keyboard shortcuts so that gesture inputs can control YouTube.

Data Collection

Data was gathered using the onboard Inertial Measurement Unit (IMU) of the Arduino Nano BLE 33 Sense, which captures acceleration across three axes: X, Y, and Z. This acceleration data was sampled and subsequently processed using Edge Impulse for analysis.

Steps for Data Collection:

- **Sensor Data**: Acceleration measurements were obtained along three axes—**accX**, **accY**, and **accZ**.
- **Gestures Recorded**: Three distinct gestures were tracked: circular motion, vertical motion (up/down), and lateral motion (pan).
    - **Circle Gesture**: Represents the command to move to the next video.
    - **Up/Down Gesture**: Corresponds to the action of muting or unmuting.
    - **Pan Gesture**: Linked to the play/pause function for media control.



Avinash Baraiya / 202201211_Gesture Project PERSONAL

Target: Cortex-M4F 80MHz

DATA COLLECTED 2m 31s

TRAIN / TEST SPLIT 51% / 49%

Dataset

Training (21) Test (20)

SAMPLE NAME	LABEL	ADDED	LENGTH
idle.56qpv9fj	idle	Sep 04 2024, 14:49:58	20s
play-pause.56qprcpe	play-pause	Sep 04 2024, 14:47:50	3s
play-pause.56qpr1pg	play-pause	Sep 04 2024, 14:47:39	3s
play-pause.56qpq4h0	play-pause	Sep 04 2024, 14:47:09	3s
play-pause.56qppprp	play-pause	Sep 04 2024, 14:46:58	3s
play-pause.56qppfgp	play-pause	Sep 04 2024, 14:46:47	3s
play-pause.56qpp4ug	play-pause	Sep 04 2024, 14:46:36	3s
play-pause.56qpopn3	play-pause	Sep 04 2024, 14:46:25	3s
backward.56qpnrsh	backward	Sep 04 2024, 14:45:54	3s
backward.56qpnejb	backward	Sep 04 2024, 14:45:41	3s
backward.56qpn2rb	backward	Sep 04 2024, 14:45:29	3s

Collect data

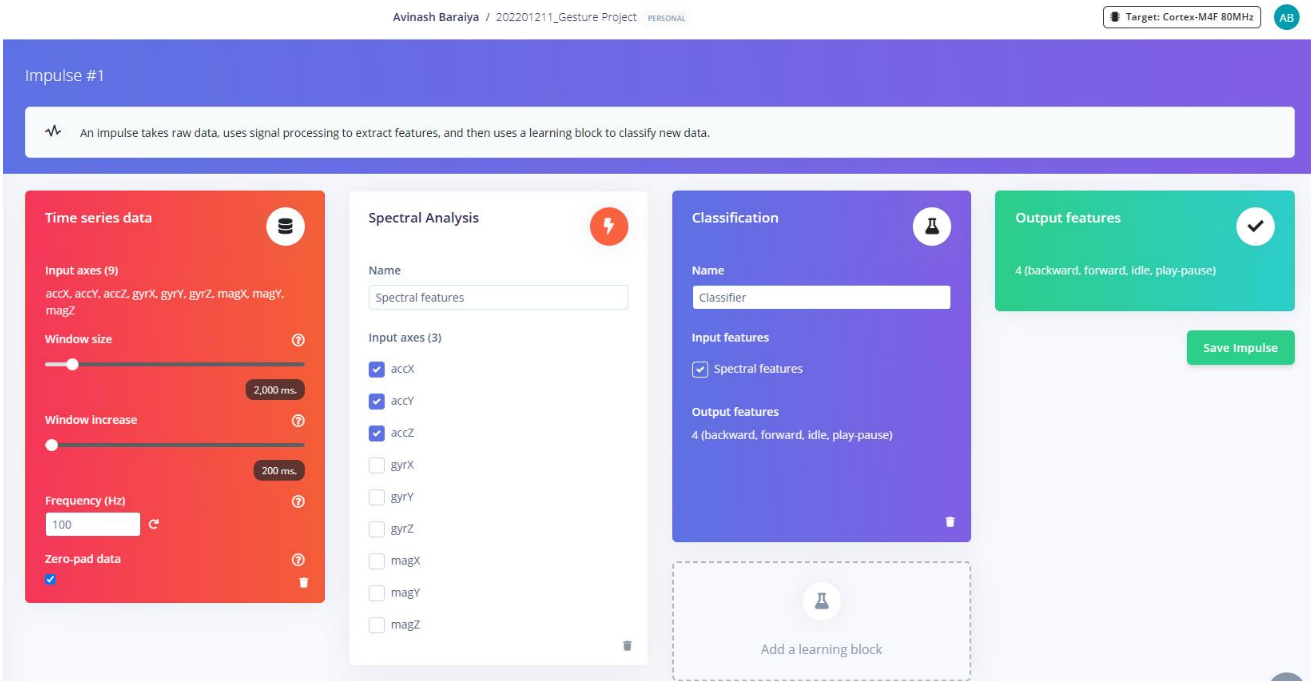
Connect a device to start building your dataset.

RAW DATA

Click on a sample to load...

Figure 1 : Data Acquisition

Spectral Analysis



Avinash Baraiya / 202201211_Gesture Project PERSONAL

Target: Cortex-M4F 80MHz

Impulse #1

An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.

Time series data

Input axes (9)

accX, accY, accZ, gyrX, gyrY, gyrZ, magX, magY, magZ

Window size: 2,000 ms.

Window increase: 200 ms.

Frequency (Hz): 100

Zero-pad data ☑

Spectral Analysis

Name: Spectral features

Input axes (3)

☑ accX
☑ accY
☑ accZ
□ gyrX
□ gyrY
□ gyrZ
□ magX
□ magY
□ magZ

Classification

Name: Classifier

Input features

☑ Spectral features

Output features

4 (backward, forward, idle, play-pause)

Add a learning block

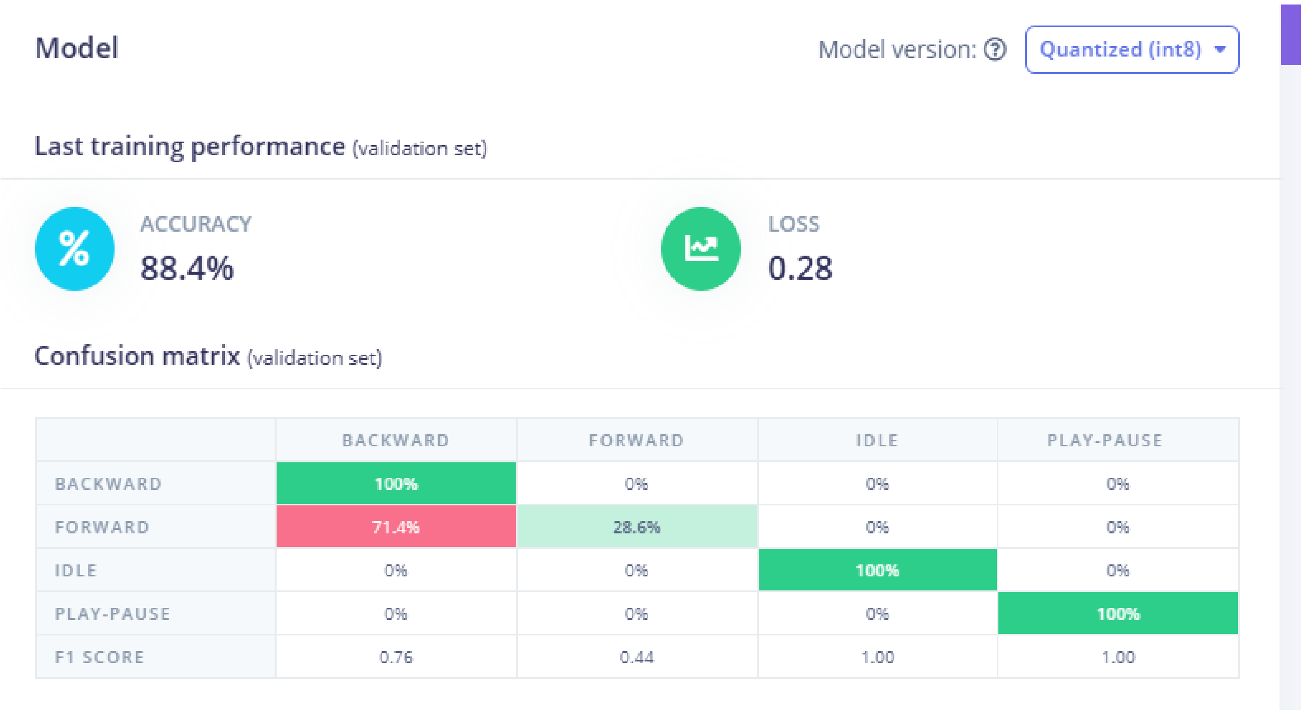
Output features

4 (backward, forward, idle, play-pause)

Save Impulse

Figure 2 : Create Impulse

Classification



Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 88.4%

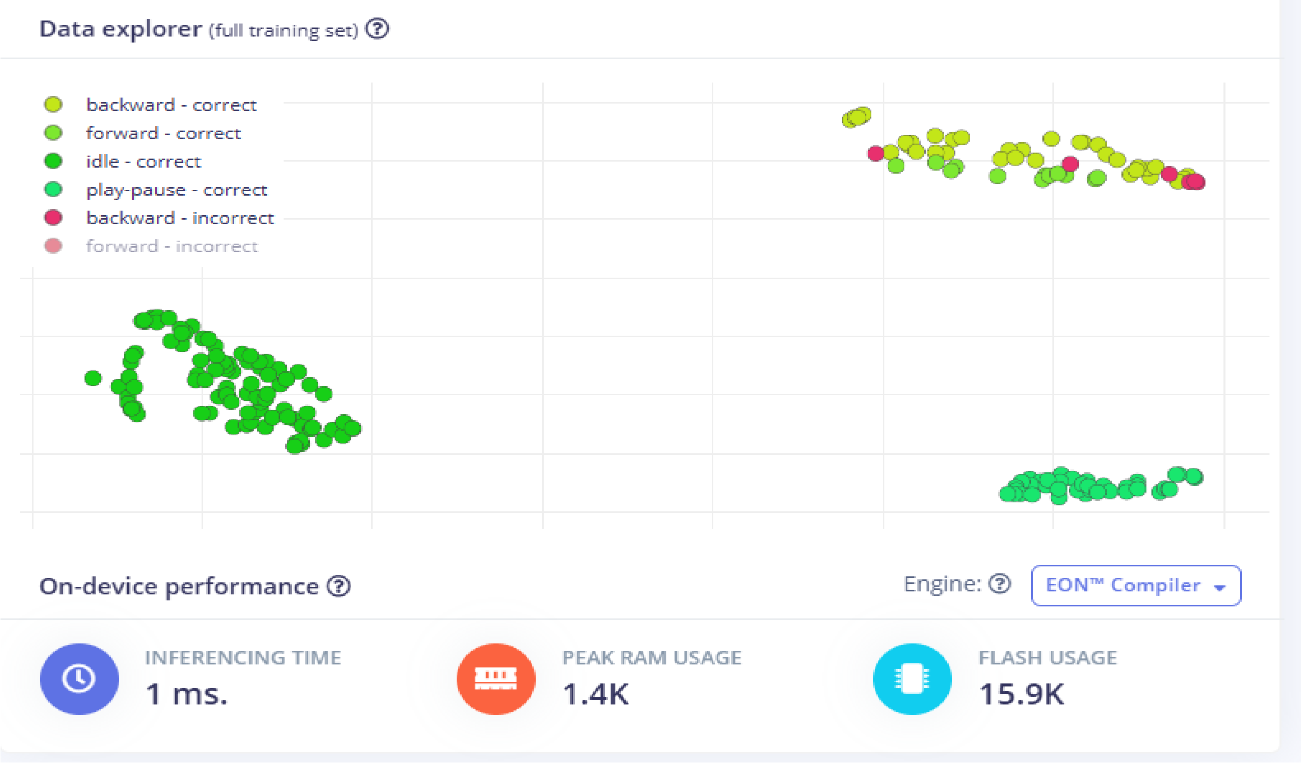
LOSS 0.28

Confusion matrix (validation set)

	BACKWARD	FORWARD	IDLE	PLAY-PAUSE
BACKWARD	100%	0%	0%	0%
FORWARD	71.4%	28.6%	0%	0%
IDLE	0%	0%	100%	0%
PLAY-PAUSE	0%	0%	0%	100%
F1 SCORE	0.76	0.44	1.00	1.00

Testing and Results

Once the model was trained, we tested its performance by feeding it new gesture data and verifying if it correctly classified the actions. The model's performance was satisfactory for all the gestures with minimal errors.



Data explorer (full training set) ⓘ

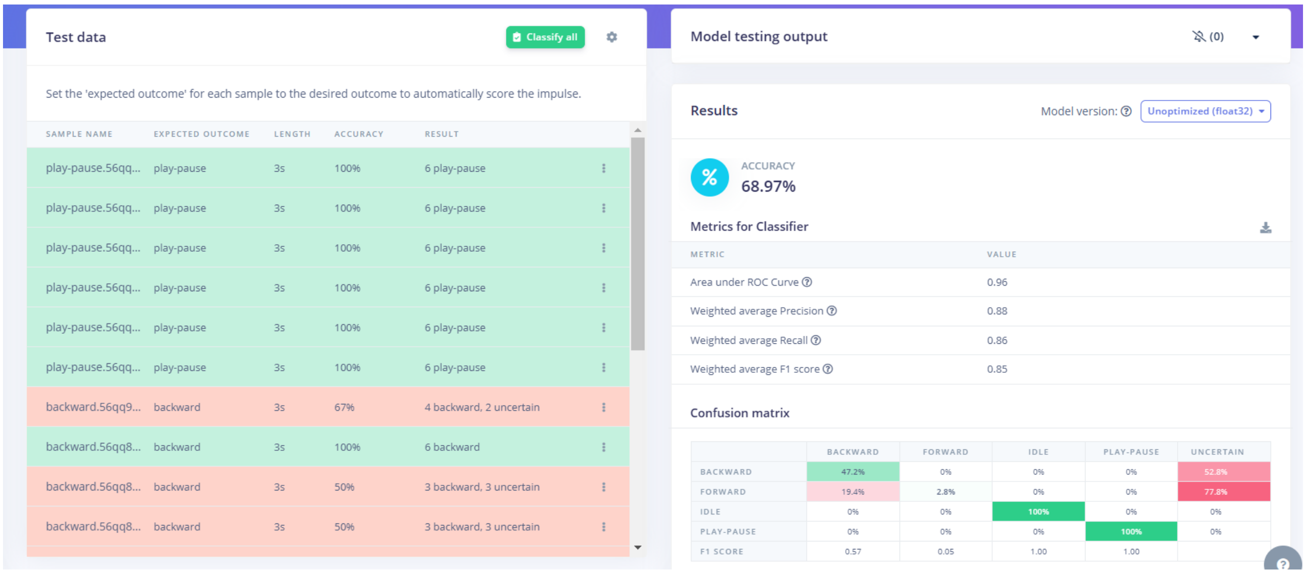
On-device performance ⓘ

Engine: ⓘ EON™ Compiler

INFERENCING TIME 1 ms.

PEAK RAM USAGE 1.4K

FLASH USAGE 15.9K



Test data

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

SAMPLE NAME	EXPECTED OUTCOME	LENGTH	ACCURACY	RESULT
play-pause.56qq...	play-pause	3s	100%	6 play-pause
play-pause.56qq...	play-pause	3s	100%	6 play-pause
play-pause.56qq...	play-pause	3s	100%	6 play-pause
play-pause.56qq...	play-pause	3s	100%	6 play-pause
play-pause.56qq...	play-pause	3s	100%	6 play-pause
play-pause.56qq...	play-pause	3s	100%	6 play-pause
backward.56qq9...	backward	3s	67%	4 backward, 2 uncertain
backward.56qq8...	backward	3s	100%	6 backward
backward.56qq8...	backward	3s	50%	3 backward, 3 uncertain
backward.56qq8...	backward	3s	50%	3 backward, 3 uncertain

Model testing output

Results

Model version: ⓘ Unoptimized (float32)

ACCURACY 68.97%

Metrics for Classifier

METRIC	VALUE
Area under ROC Curve ⓘ	0.96
Weighted average Precision ⓘ	0.88
Weighted average Recall ⓘ	0.86
Weighted average F1 score ⓘ	0.85

Confusion matrix

	BACKWARD	FORWARD	IDLE	PLAY-PAUSE	UNCERTAIN
BACKWARD	47.2%	0%	0%	0%	52.8%
FORWARD	19.4%	2.8%	0%	0%	77.8%
IDLE	0%	0%	100%	0%	0%
PLAY-PAUSE	0%	0%	0%	100%	0%
F1 SCORE	0.57	0.05	1.00	1.00	

Code Implementation

- Reads the gesture commands from the Serial port.
- Uses the PyAutoGUI library to simulate YouTube keyboard controls:
    - "next" → right arrow key (next video)
    - "mute" → 'm' key (mute/unmute)
    - "play" → 'k' key (play/pause)

```python
# if you don't have pyautogui & pyserial installed
# pip install pyserial pyautogui

# pyautogui is used for control of the mouse and keyboard, and other GUI automation tasks
# pyserial is a python library used for working with serial ports
# time is used to create time related task in the program for delaying or early activation of certain function

import serial
import time
import pyautogui

# Set thresholds for gesture detection or confidence threshold to make sure your desired gesture is working
CIRCLE_THRESHOLD = 0.7
CROSS_THRESHOLD = 0.7
PAN_THRESHOLD = 0.7

def open_serial_connection():
    try:
        ser = serial.Serial('COM3', 115200) # change COM Port as per system is showing or open arduino IDE to check which COM Port is Used
        print("Connected to the serial port")
        time.sleep(2)  # Wait for the serial connection to initialize
        return ser
    except serial.SerialException as e:
        print(f"Error connecting to serial port: {e}")
        return None

def classify_gesture(incoming):
    # Extracting gesture and confidence from the incoming data
    if 'circle' in incoming:
        confidence = float(incoming.split(':')[1].strip())
        if confidence > CIRCLE_THRESHOLD:
            print(f"Circle gesture detected with confidence {confidence}. Pressing 'm' key")
            pyautogui.press('m')
    elif 'cross' in incoming:
        confidence = float(incoming.split(':')[1].strip())
        if confidence > CROSS_THRESHOLD:
            print(f"Cross gesture detected with confidence {confidence}. Pressing
'k' key")
            pyautogui.press('k')
    elif 'pan' in incoming:
        confidence = float(incoming.split(':')[1].strip())
        if confidence > PAN_THRESHOLD:
            print(f"Pan gesture detected with confidence {confidence}. Pressing
'Shift + n' key")
            pyautogui.hotkey('shift', 'n')

def main():
    ser = open_serial_connection()
    if ser is None:
        return  # Exit if unable to connect

    try:
        while True:
            try:
                incoming = ser.readline().decode('utf-8').strip()
                print(f"Received: {incoming}")

                # Look for predictions in the incoming data
                if "Predictions" in incoming:
                    # Continue reading lines until we get the classification result
                    for _ in range(3):
                        incoming = ser.readline().decode('utf-8').strip()
                        classify_gesture(incoming)

            except serial.SerialException as e:
                print(f"Error reading from serial port: {e}")
                break
            except UnicodeDecodeError as e:
                print(f"Decoding error: {e}")
            except Exception as e:
                print(f"Unexpected error: {e}")

    finally:
        if ser is not None and ser.is_open:
            ser.close()
            print("Serial port closed.")

if __name__ == "__main__":
    main()
```

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.
 *
 */

/* Includes ---------------------------------------------------------------- */
#include <a202201211_Gesture_Project_inferencing.h>
#include <Arduino_LSM9DS1.h> //Click here to get the library:
https://www.arduino.cc/reference/en/libraries/arduino_lsm9ds1/

/* Constant defines -------------------------------------------------------- */
#define CONVERT_G_TO_MS2    9.80665f
#define MAX_ACCEPTED_RANGE  2.0f        // starting 03/2022, models are generated
setting range to +-2, but this example use Arudino library which set range to +-4g.
If you are using an older model, ignore this value and use 4.0f instead

/*
 ** 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.
 */

/* Private variables ------------------------------------------------------- */
```

```
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");

    if (!IMU.begin()) {
        ei_printf("Failed to initialize IMU!\r\n");
    }
    else {
        ei_printf("IMU initialized\r\n");
    }

    if (EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME != 3) {
        ei_printf("ERR: EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME should be equal to 3
(the 3 sensor axes)\n");
        return;
    }
}

/**
 * @brief Return the sign of the number
 *
 * @param number
 * @return int 1 if positive (or 0) -1 if negative
 */
float ei_get_sign(float number) {
    return (number >= 0.0) ? 1.0 : -1.0;
}

/**
* @brief      Get data and run inferencing
*
* @param[in]  debug  Get debug info if true
*/
void loop()
{
    ei_printf("\nStarting inferencing in 2 seconds...\n");
    delay(2000);

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

    // Allocate a buffer here for the values we'll read from the IMU
    float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };

    for (size_t ix = 0; ix < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; ix += 3) {
        // Determine the next tick (and then sleep later)
        uint64_t next_tick = micros() + (EI_CLASSIFIER_INTERVAL_MS * 1000);

        IMU.readAcceleration(buffer[ix], buffer[ix + 1], buffer[ix + 2]);

        for (int i = 0; i < 3; i++) {
            if (fabs(buffer[ix + i]) > MAX_ACCEPTED_RANGE) {
                buffer[ix + i] = ei_get_sign(buffer[ix + i]) * MAX_ACCEPTED_RANGE;
            }
        }

        buffer[ix + 0] *= CONVERT_G_TO_MS2;
        buffer[ix + 1] *= CONVERT_G_TO_MS2;
        buffer[ix + 2] *= CONVERT_G_TO_MS2;

        delayMicroseconds(next_tick - micros());
    }

    // Turn the raw buffer in a signal which we can the classify
    signal_t signal;
    int err = numpy::signal_from_buffer(buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE,
&signal);
    if (err != 0) {
        ei_printf("Failed to create signal from buffer (%d)\n", err);
        return;
    }

    // Run the classifier
    ei_impulse_result_t result = { 0 };

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

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

Practical Working Video Link

https://youtu.be/ExCY1Zeibpg?si=sdKgYLfXkwahvYO1