

Edge Computing Laboratory

Lab Assignment 6

Name: Gaurav Gadekar

Class: TY AIEC Batch C

Enrollment No: MITU23BTCSD120

Roll No: D2233120

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:

Create an Account and New Project:

- Sign up for an Edge Impulse account.
- Create a new project from the dashboard.

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.

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.

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.

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.

Train the Model:

- Click on the 'Start training' button to train your machine learning model with the collected data.

Test the Model:

- Once the model is trained, you can test its performance with new data in the 'Model Testing' tab.

Deploy the Model:

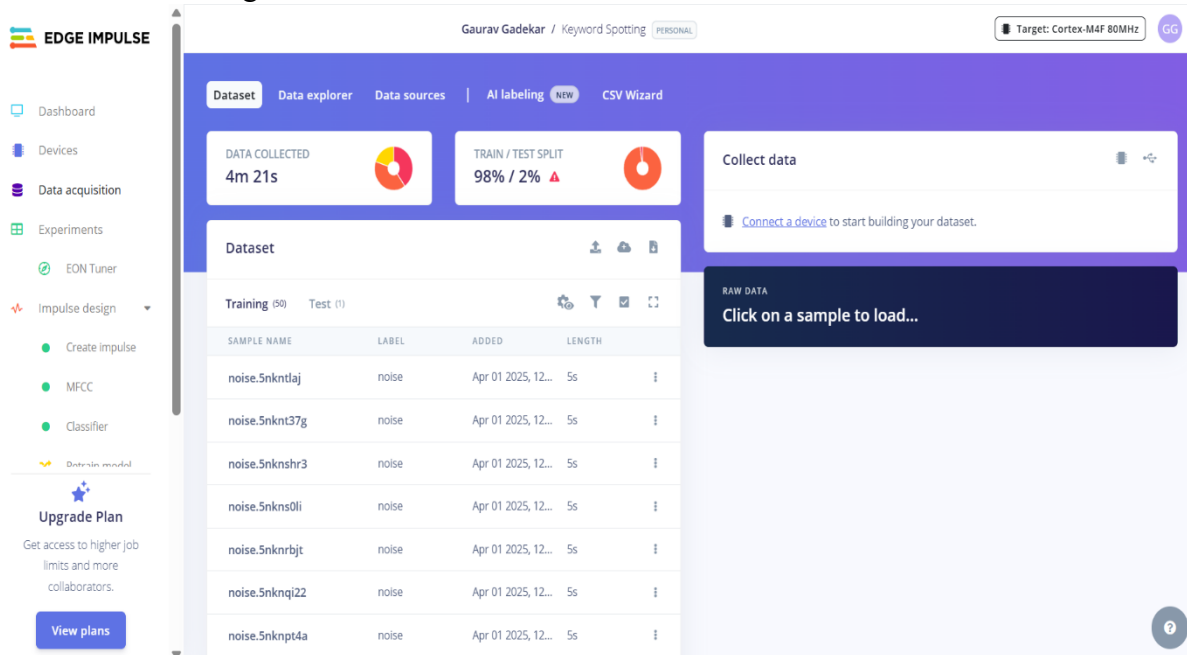
- Go to the 'Deployment' tab.
- Select the deployment method that suits your edge device (e.g., Arduino library, WebAssembly, container, etc.).
- Follow the instructions to deploy the model to your device.

Run Inference:

- With the model deployed, run inference on the edge device to see it classifying data in realtime.
- Monitor:
- You can monitor the performance of your device through the Edge Impulse studio.

Screenshots:

1. Dataset Image

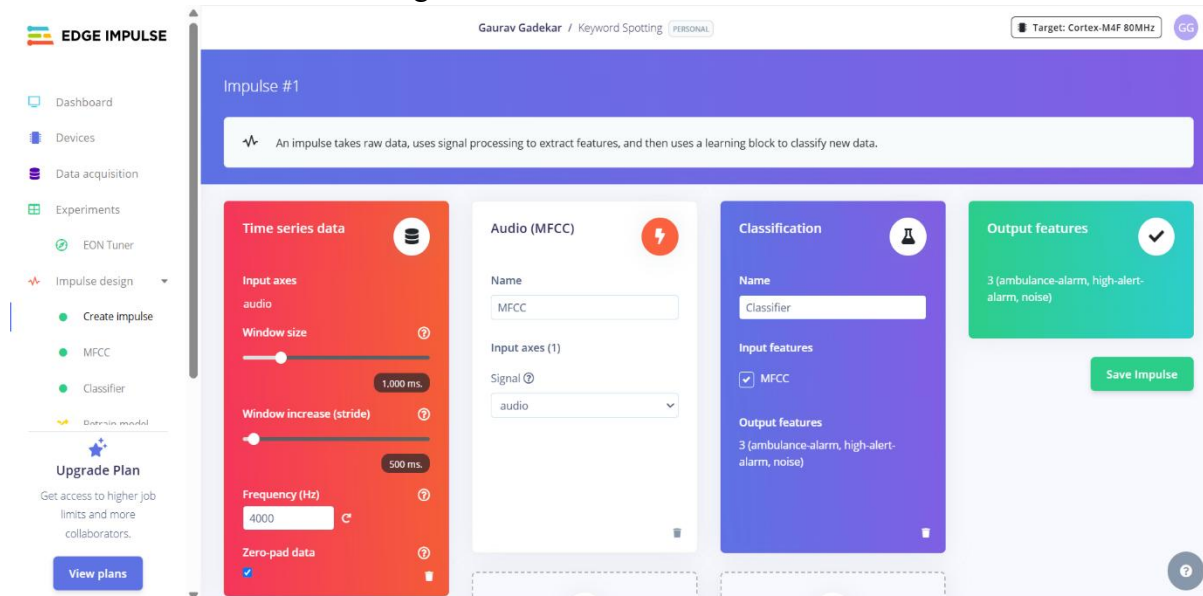


The screenshot shows the 'Dataset' page in the EDGE IMPULSE interface. The user is logged in as 'Gaurav Gadekar' and is working on a 'Keyword Spotting' project. The target device is 'Cortex-M4F 80MHz'. The page has a sidebar with navigation options: Dashboard, Devices, Data acquisition, Experiments, EON Tuner, and Impulse design. The main content area has tabs for 'Dataset', 'Data explorer', 'Data sources', 'AI labeling', and 'CSV Wizard'. The 'Dataset' tab is active, showing a 'DATA COLLECTED' status of '4m 21s' and a 'TRAIN / TEST SPLIT' of '98% / 2%'. A 'Collect data' button is present, with a link to 'Connect a device to start building your dataset.' Below this, a table lists the dataset samples:

SAMPLE NAME	LABEL	ADDED	LENGTH
noise.5nknltaj	noise	Apr 01 2025, 12...	5s
noise.5nknlt37g	noise	Apr 01 2025, 12...	5s
noise.5nknshr3	noise	Apr 01 2025, 12...	5s
noise.5nknsl0li	noise	Apr 01 2025, 12...	5s
noise.5nknrbjt	noise	Apr 01 2025, 12...	5s
noise.5nknqj22	noise	Apr 01 2025, 12...	5s
noise.5nknpt4a	noise	Apr 01 2025, 12...	5s

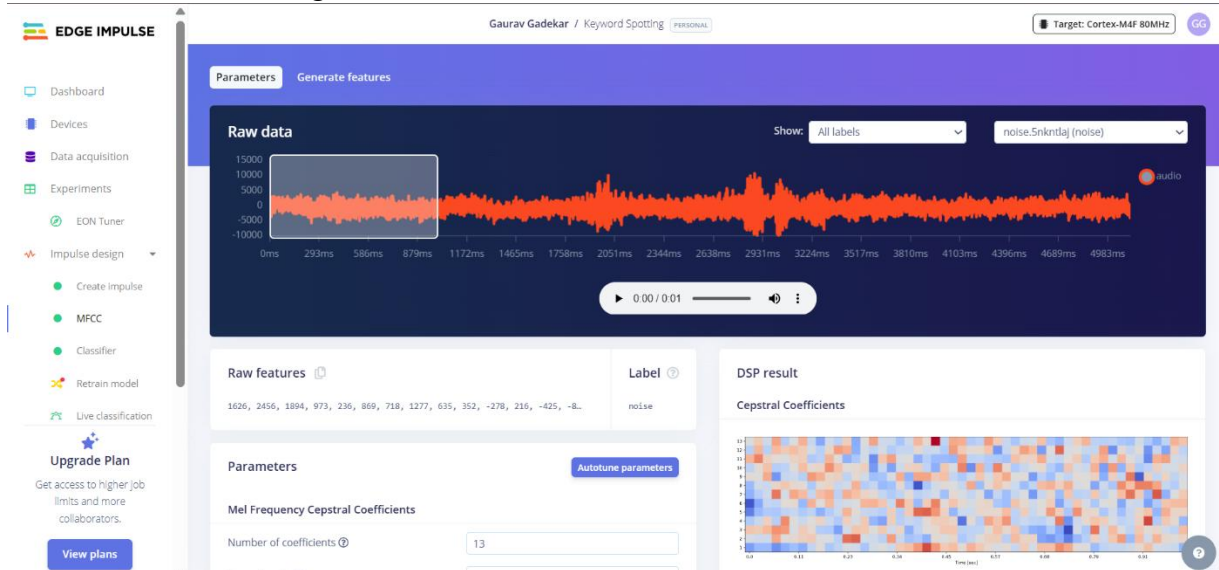
On the right, there is a 'RAW DATA' section with a button to 'Click on a sample to load...'. An 'Upgrade Plan' button is visible in the sidebar.

2. Feature extraction - Image



The screenshot shows the 'Feature extraction - Image' page in the EDGE IMPULSE interface. The user is logged in as 'Gaurav Gadekar' and is working on a 'Keyword Spotting' project. The target device is 'Cortex-M4F 80MHz'. The page has a sidebar with navigation options: Dashboard, Devices, Data acquisition, Experiments, EON Tuner, and Impulse design. The main content area has a header 'Impulse #1' and a description: 'An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.' Below this, there are four main sections: 'Time series data', 'Audio (MFCC)', 'Classification', and 'Output features'. The 'Time series data' section has settings for 'Input axes' (audio), 'Window size' (1,000 ms), 'Window increase (stride)' (500 ms), 'Frequency (Hz)' (4000), and 'Zero-pad data' (checked). The 'Audio (MFCC)' section has settings for 'Name' (MFCC), 'Input axes (1)' (audio), and 'Signal' (audio). The 'Classification' section has settings for 'Name' (Classifier), 'Input features' (MFCC), and 'Output features' (3 (ambulance-alarm, high-alert-alarm, noise)). The 'Output features' section has a 'Save Impulse' button. An 'Upgrade Plan' button is visible in the sidebar.

3. Validation Result – Image



4. Copy the code of Arduino Sketch

```
/* 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 <Gaurav-gadekar-Keyword-Spotting.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
/**
 * When data is collected by the Edge Impulse Arduino Nano 33 BLE Sense * firmware, it is limited
 * to a 2G range. If the model was created with a * different sample range, modify this constant
 * to match the input values. * See https://github.com/edgeimpulse/firmware-arduino-nano-33-
 * blesense/blob/master/src/sensors/ei_lsm9ds1.cpp
 * for more information.
 */
#define MAX_ACCEPTED_RANGE 2.0f
// -DEI_CLASSIFIER_ALLOCATION_STATIC
```

```

/*
** 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
** `https://support.arduino.cc/hc/en-us/articles/360012076960-Where-are-theinstalled-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
static uint32_t run_inference_every_ms = 200; static rtos::Thread
inference_thread(osPriorityLow); static float
buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 }; static float
inference_buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE];

/* Forward declaration */ void
run_inference_background();

/**
* @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;
    } inference_thread.start(mbed::callback(&run_inference_background)); }

/**
* @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      Run inferencing in the background.
*/ void run_inference_background()

```

```

{
    // wait until we have a full buffer          delay((EI_CLASSIFIER_INTERVAL_MS *
EI_CLASSIFIER_RAW_SAMPLE_COUNT) + 100);
    // This is a structure that smoothens the output result
    // With the default settings 70% of readings should be the same before classifying.
    ei_classifier_smooth_t smooth;    ei_classifier_smooth_init(&smooth, 10 /* no.
of readings */, 7 /* min. readings the same */, 0.8 /* min. confidence */, 0.3 /* max
anomaly */);    while (1) {
        // copy the buffer          memcpy(inference_buffer, buffer,
EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE * sizeof(float));

        // Turn the raw buffer in a signal which we can the classify          signal_t
signal;
        int err = numpy::signal_from_buffer(inference_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(": ");

        // ei_classifier_smooth_update yields the predicted label          const char
*prediction = ei_classifier_smooth_update(&smooth, &result);    ei_printf("%s ",
prediction);    // print the cumulative results          ei_printf(" [ ");
        for (size_t ix = 0; ix < smooth.count_size; ix++) {
ei_printf("%u", smooth.count[ix]);    if (ix !=
smooth.count_size + 1) {
ei_printf(", ");
        }
    }
    else {
ei_printf(" ");
    }
    ei_printf("]\n");
        delay(run_inference_every_ms);
    }    ei_classifier_smooth_free(&smooth);
}

/**
 * @brief      Get data and run inferencing
 *
 * @param[in]  debug  Get debug info if true
 */ void
loop() {
    while (1) {
        // Determine the next tick (and then sleep later)          uint64_t next_tick =
micros() + (EI_CLASSIFIER_INTERVAL_MS * 1000);
        // roll the buffer -3 points so we can overwrite the last one          numpy::roll(buffer,
EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, -3);

```

```

        // read to the end of the buffer          IMU.readAcceleration(
buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3],
buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 2],
buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 1]
        );          for (int i = 0; i < 3;
i++) {
            if (fabs(buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3 + i]) >
MAX_ACCEPTED_RANGE) {
                buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3 +
i] = ei_get_sign(buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3 + i]) *
MAX_ACCEPTED_RANGE;
            }
        }

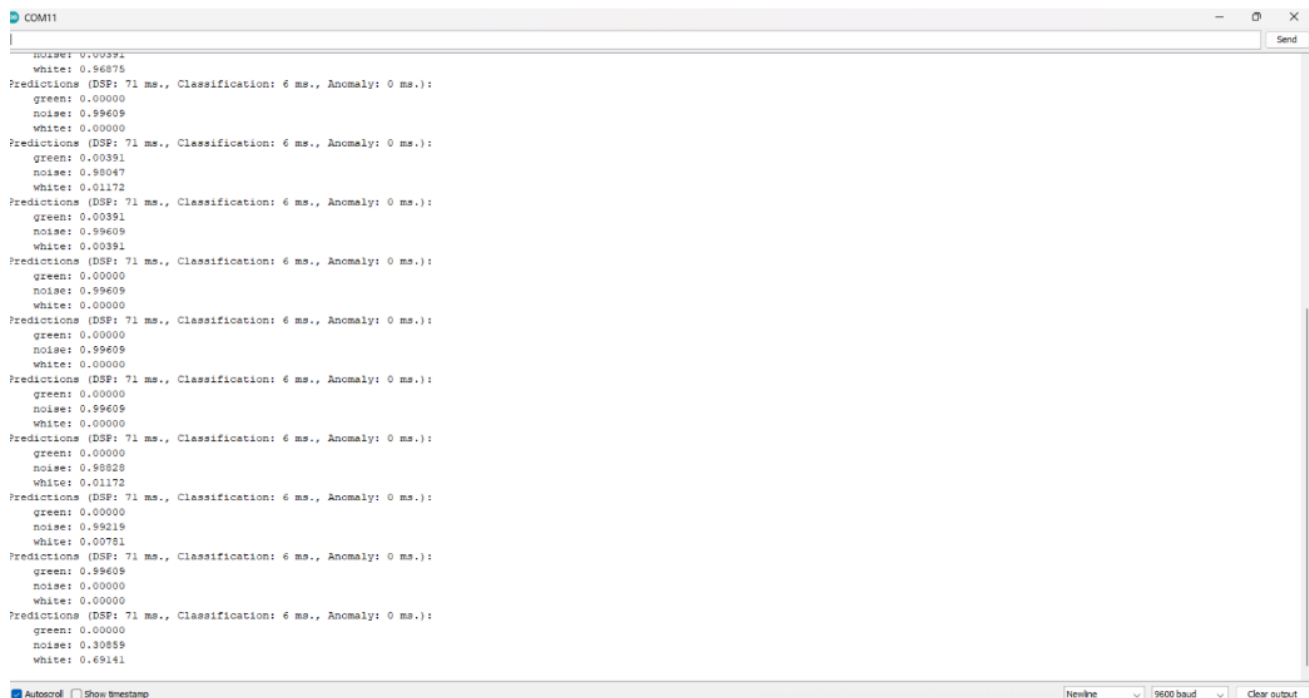
        buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3] *= CONVERT_G_TO_MS2;
        buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 2] *= CONVERT_G_TO_MS2;
        buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 1] *= CONVERT_G_TO_MS2;

        // and wait for next tick          uint64_t time_to_wait =
next_tick - micros();
delay((int)floor((float)time_to_wait / 1000.0f));
delayMicroseconds(time_to_wait % 1000);
    }
}

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

```

5. Screen shot of Arduino Terminal – Result



The screenshot shows an Arduino IDE terminal window titled 'COM11'. The terminal displays a series of sensor readings and predictions. Each line of output follows a similar pattern: 'Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):' followed by three lines of data: 'green: 0.00000', 'noise: 0.99609', and 'white: 0.00000'. The 'noise' value is consistently 0.99609, while the 'green' and 'white' values are consistently 0.00000. The terminal also shows a 'Send' button in the top right corner and a status bar at the bottom with 'Autoscroll' checked, 'Show timestamp' unchecked, and a baud rate of 9600.

```

noise: 0.00000
white: 0.96875
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
green: 0.00000
noise: 0.99609
white: 0.00000
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
green: 0.00391
noise: 0.98047
white: 0.01172
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
green: 0.00391
noise: 0.99609
white: 0.00391
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
green: 0.00000
noise: 0.99609
white: 0.00000
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
green: 0.00000
noise: 0.99609
white: 0.00000
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
green: 0.00000
noise: 0.99609
white: 0.00000
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
green: 0.00000
noise: 0.98828
white: 0.01172
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
green: 0.00000
noise: 0.99219
white: 0.00781
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
green: 0.99609
noise: 0.00000
white: 0.00000
Predictions (DSP: 71 ms., Classification: 6 ms., Anomaly: 0 ms.):
green: 0.00000
noise: 0.30859
white: 0.69141

```