Preparation:

1. Get Started with Wio Terminal
2. Connect Wio Terminal with Edge Impulse
3. Add some necessary Arduino libraries

Get Started with Wio Terminal

Step 1. You need to Install Arduino Software.

Download Arduino IDE

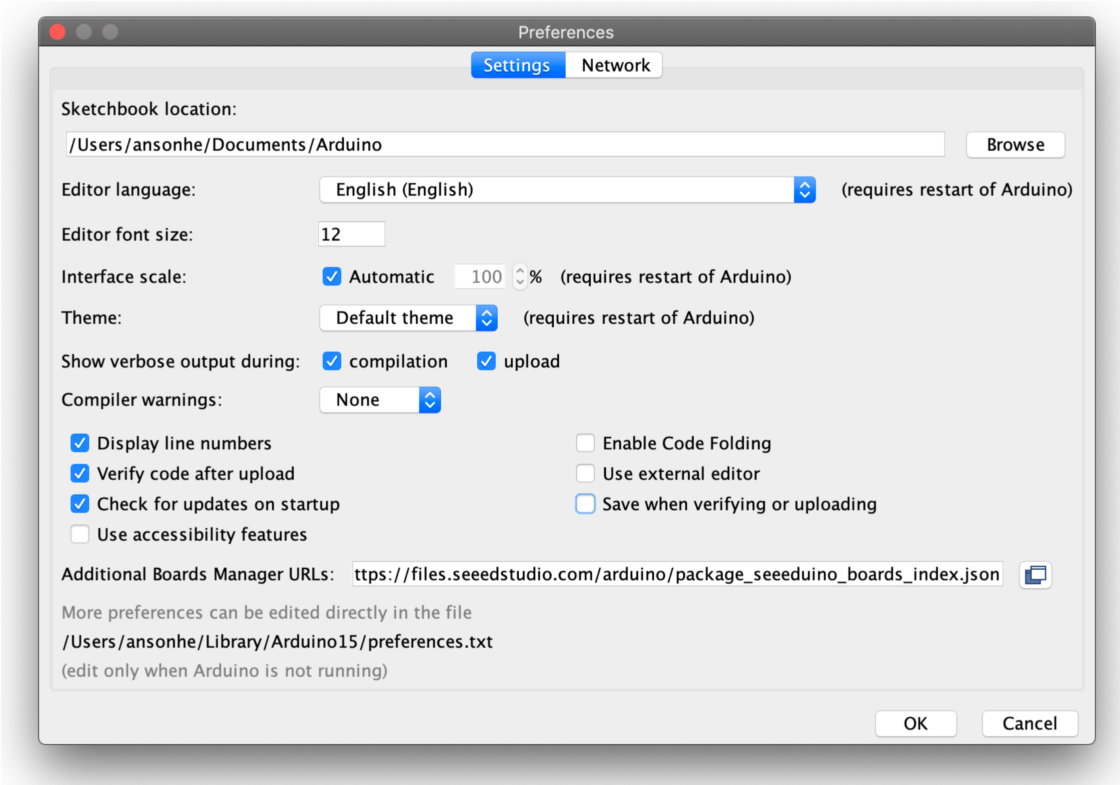
Launch the Arduino application

Double-click the Arduino IDE application you have previously downloaded.

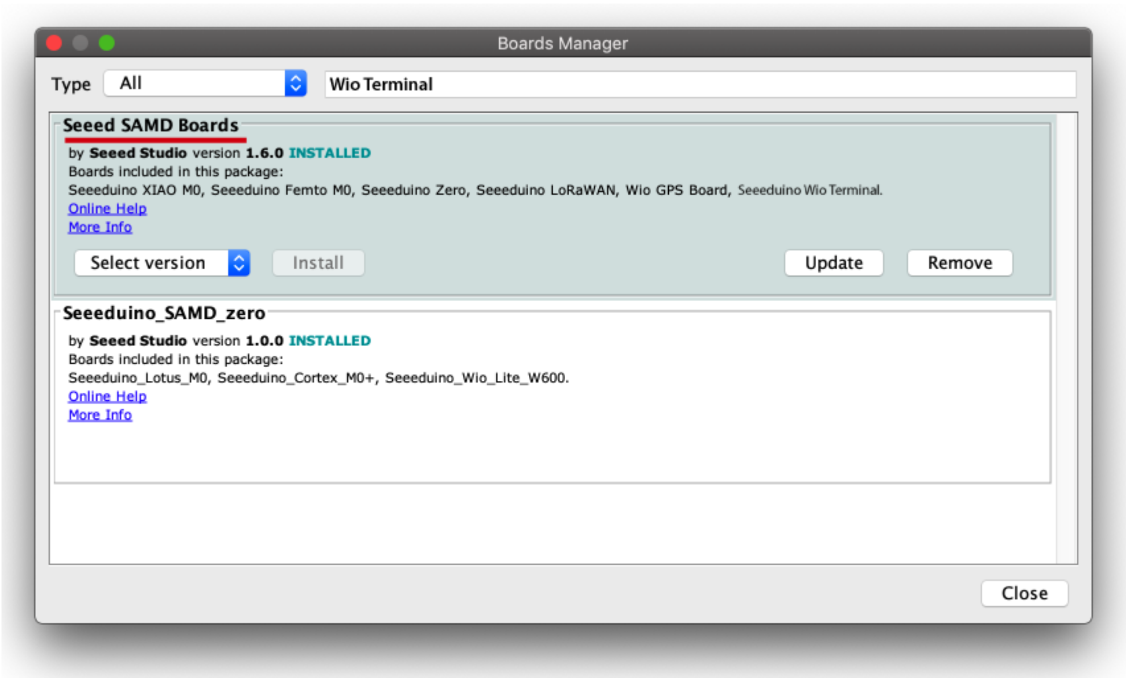
Step 2. Add the Wio Terminal Board Library

1. Open your Arduino IDE, click on **File** > **Preferences**, and copy the below URL to **Additional Boards Manager URLs**:

```
https://files.seeedstudio.com/arduino/package_seeeduino_boards_index.json
```



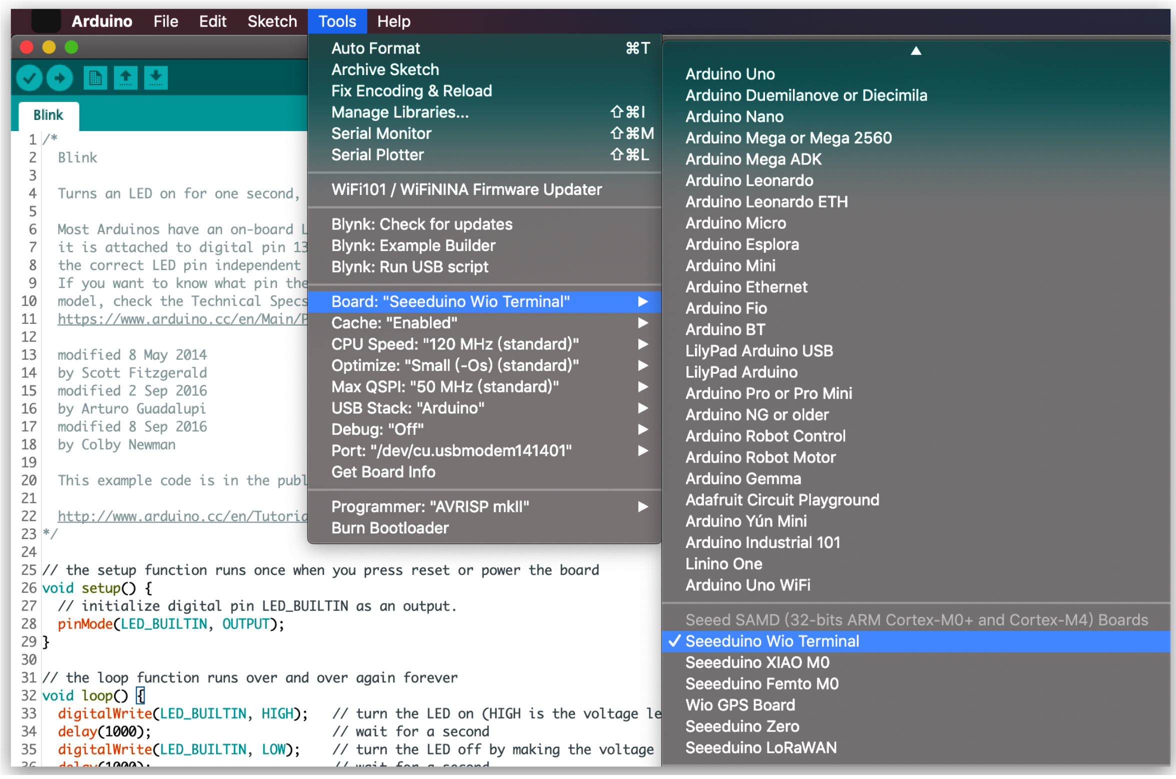
2. Click on **Tools** > **Board** > **Board Manager** and Search **Wio Terminal** in the Boards Manager.



The keyword is Wio Terminal

Step 3. Select your board and port

You'll need to select the entry in the **Tools > Board** menu that corresponds to your Arduino. Selecting the **Wio Terminal**.

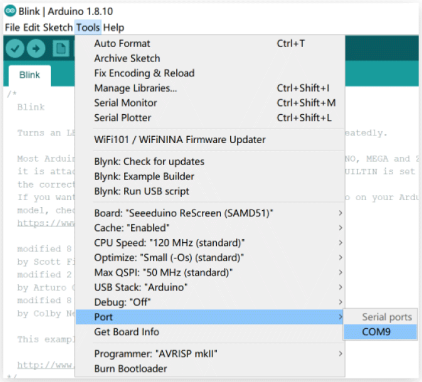


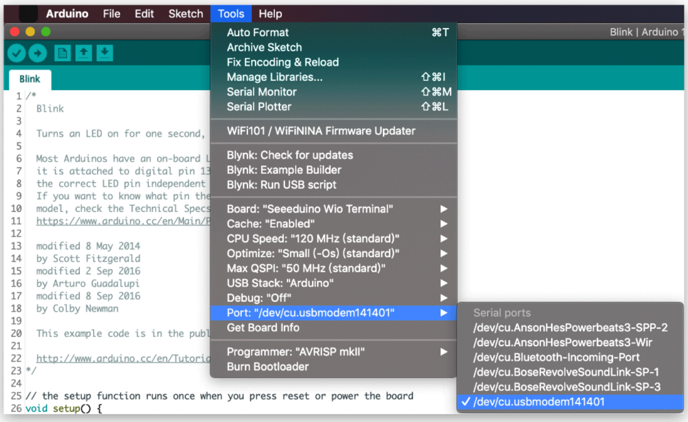
Choose the right board

Select the serial device of the Wio Terminal board from the **Tools -> Port** menu. This is likely to be COM3 or higher (**COM1** and **COM2** are usually reserved for hardware serial ports). To find out, you can disconnect your Wio Terminal board and re-open the menu; the entry that disappears should be the Arduino board. Reconnect the board and select that serial port.

Note

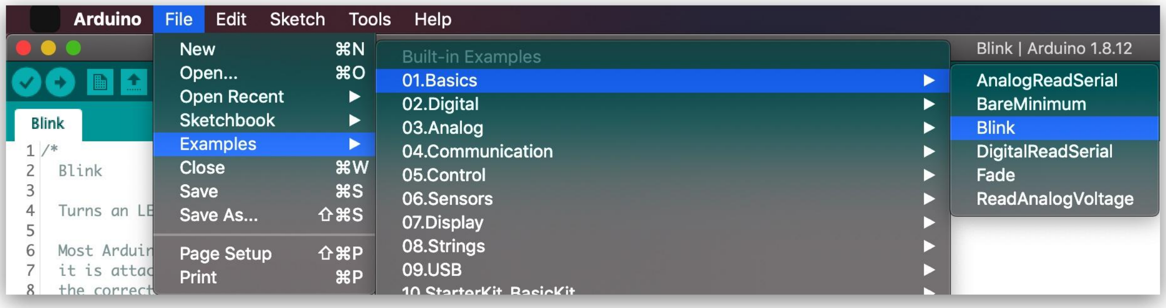
For Mac User, it will be something like `/dev/cu.usbmodem141401`





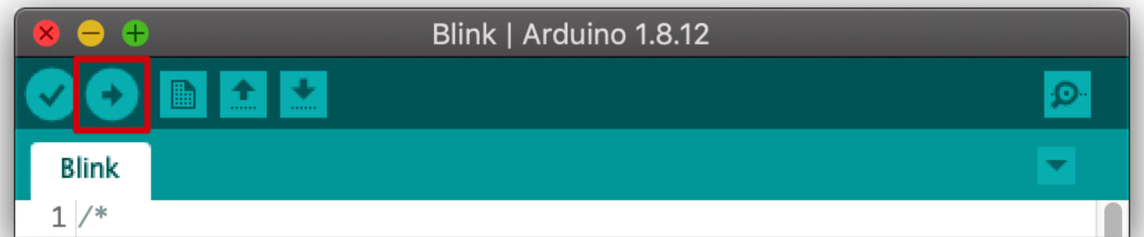
Step 4. Upload the program

Open the LED blink example sketch: File > Examples >01.Basics > Blink.



Blink Path

Now, simply click the **Upload** button in the environment. Wait a few seconds and if the upload is successful, the message "Done uploading." will appear in the status bar.



Upload the code

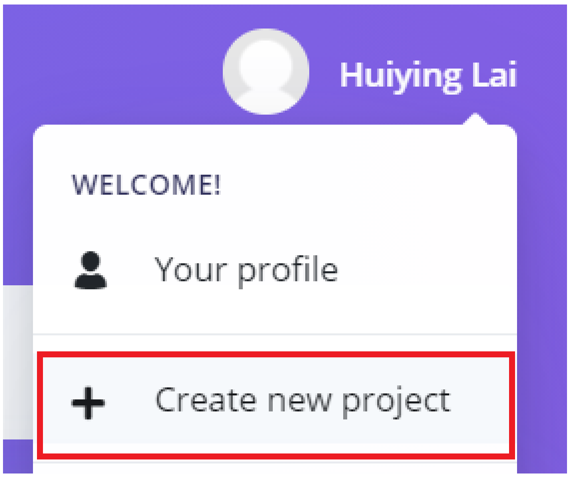
A few seconds after the upload finished, you should see the LED at the bottom of the Wio Terminal start to blink. If it does, congratulations! You've gotten Wio Terminal up-and-running. Please feel free to go through the Wiki of Wio Terminal and start building your projects!

Connect Wio Terminal with Edge Impulse

Step1: Create a new Edge Impulse project

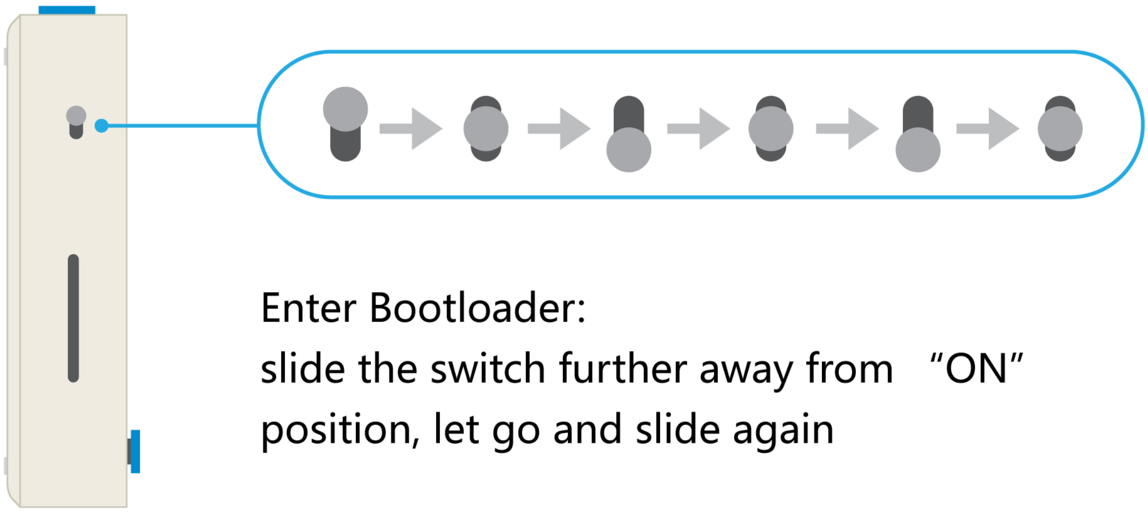
Please Open: https://www.edgeimpulse.com/

Login first, then create a new project.

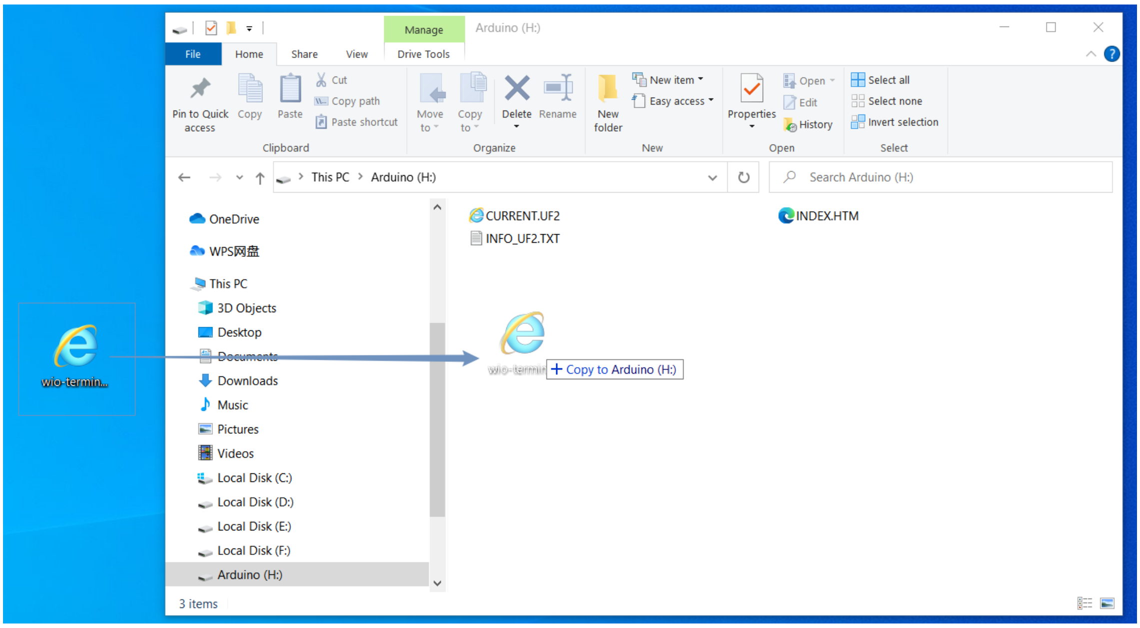


Step 2: Connect the development board to your computer

Connect Wio Terminal to your computer. Entering the bootloader mode by sliding the power switch twice quickly. For more reference, please also see here.

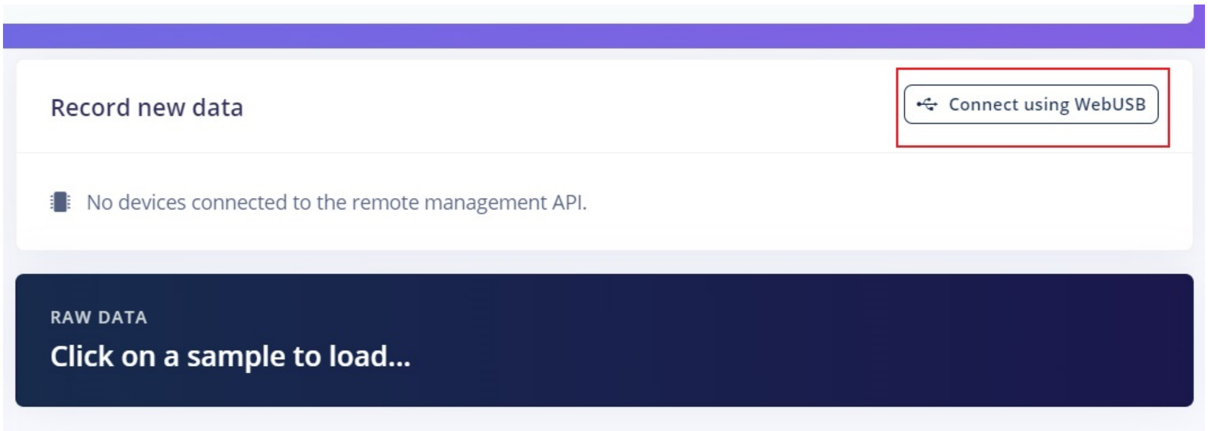


An external drive named Arduino should appear on your PC. Drag the downloaded [Edge Impulse uf2 firmware files] to the Arduino drive. Now, Edge Impulse is loaded on Wio Terminal!

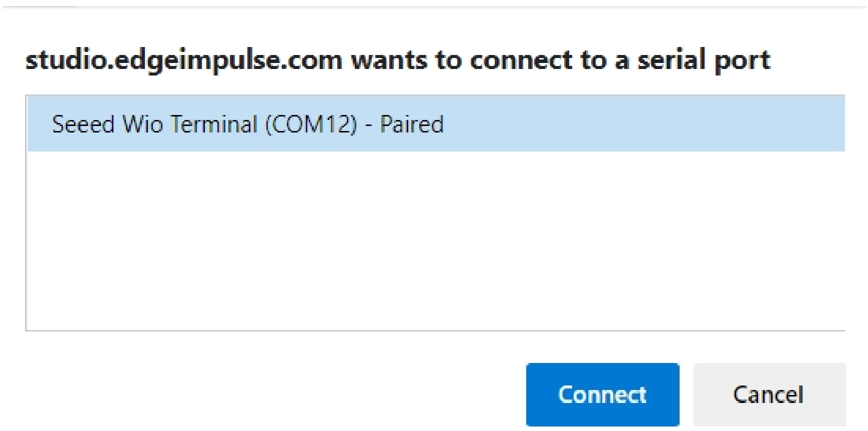


Step3: Connect using WebUSB

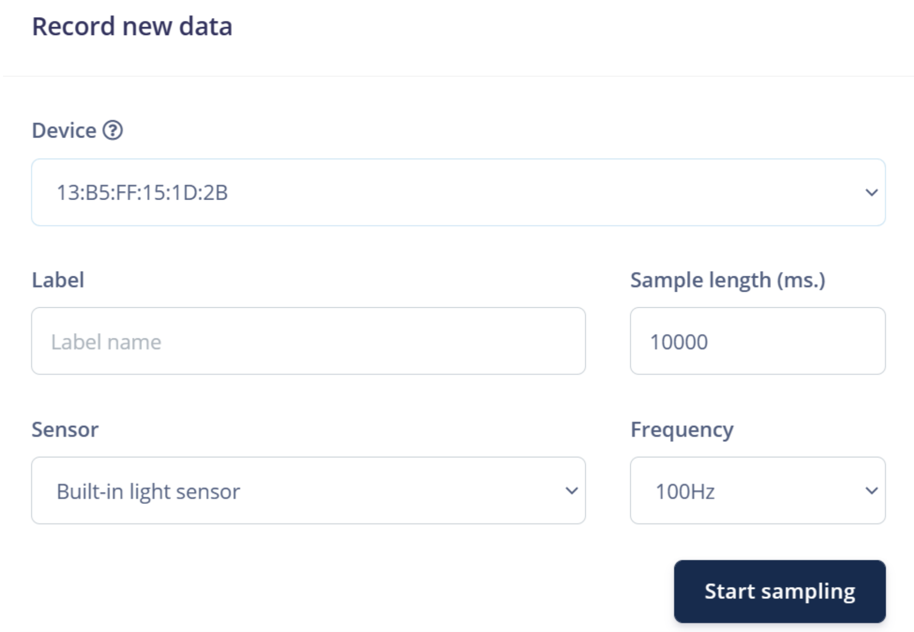
Go to your Edge Impulse project, and click the Data acquisition tab, then you can see the selection "Connect using WebUSB" on the upper right. Click it.



Then, you can see a pop-tip, select the paired serial port and "Connect" as the following picture.



Now, you have successfully connected the Wio Terminal with the Edge Impulse.



Now, all the preparations have been done, we can start our projects!

Add some necessary Arduino libraries

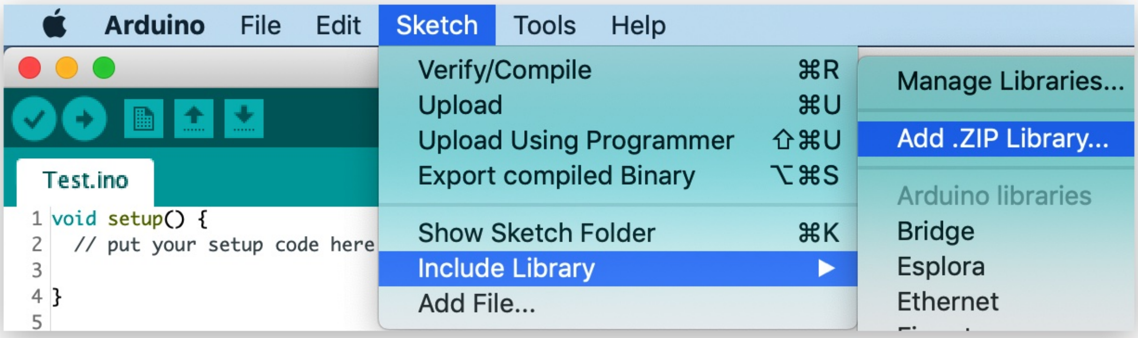
Step1:Download the library

Visit the following repositories and download the entire repo to your local drive when you are doing the corresponding Practice.

Practice 1: none

Practice 2: Seeed_Arduino_LCD 、Grove 3 Axis Digital Accelerometer
Practice 3: Seeed_Arduino_LCD
Practice 4: Seeed_Arduino_LvGL、Grove Ultrasonic Ranger、Seeed_Arduino_LCD、Seeed_Arduino_FreeRTOS
Practice 5: Grove BME280、Seeed_Arduino_LCD

Step2: Open the Arduino IDE, click sketch -> Include Library -> Add .ZIP Library, and choose the file that you have just downloaded.



Practice 1. Gesture recognition using built-in light sensor (Rock, Vulcan)

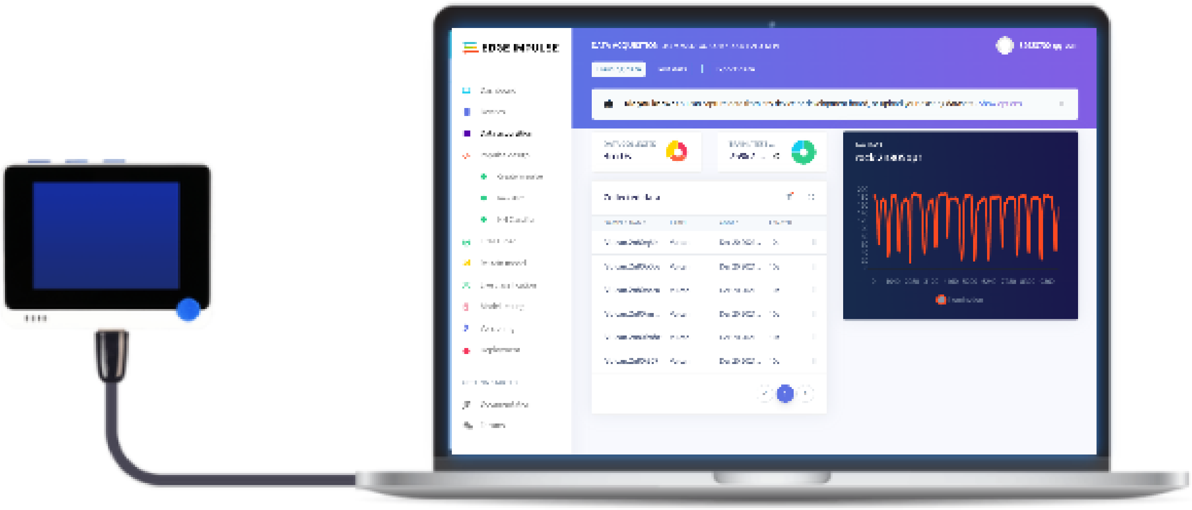
Project Overview

In this lesson, we are going to train and deploy a simple neural network for classifying Rock, Vulcan gestures with just a single light sensor.

Material Preparation

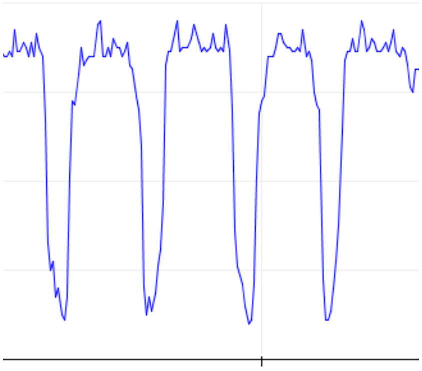
Hardware requirements: Wio Terminal

Connection method:

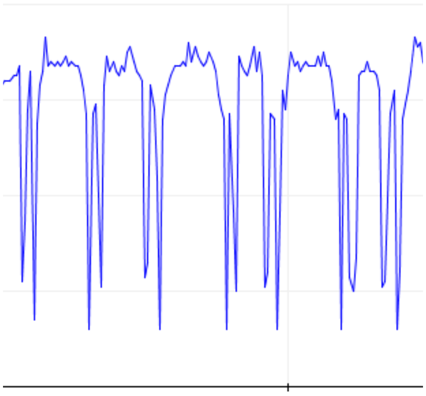


About Sensor: Built-in Light Sensor

The working principle of this project is quite trivial. Different gestures being moved above the light sensor will block a certain amount of light for certain periods of time. For example, for “Rock”, we will have high values at first (nothing above the sensor) but lower values when "Rock" passes above the sensor and then high values again. For “Vulcan”, we will have high-low-high-low-high-low-high-low values when each of the fingers in "Vulcan" passes above the sensor.



Rock



Vulcan

There is a high variance in speed and amplitude of the values from the sensor which makes a great case for using a machine learning model instead of a hand-crafted algorithm for the task.

Machine Learning Lifecycle

Please Open: https://www.edgeimpulse.com/

Data Collection

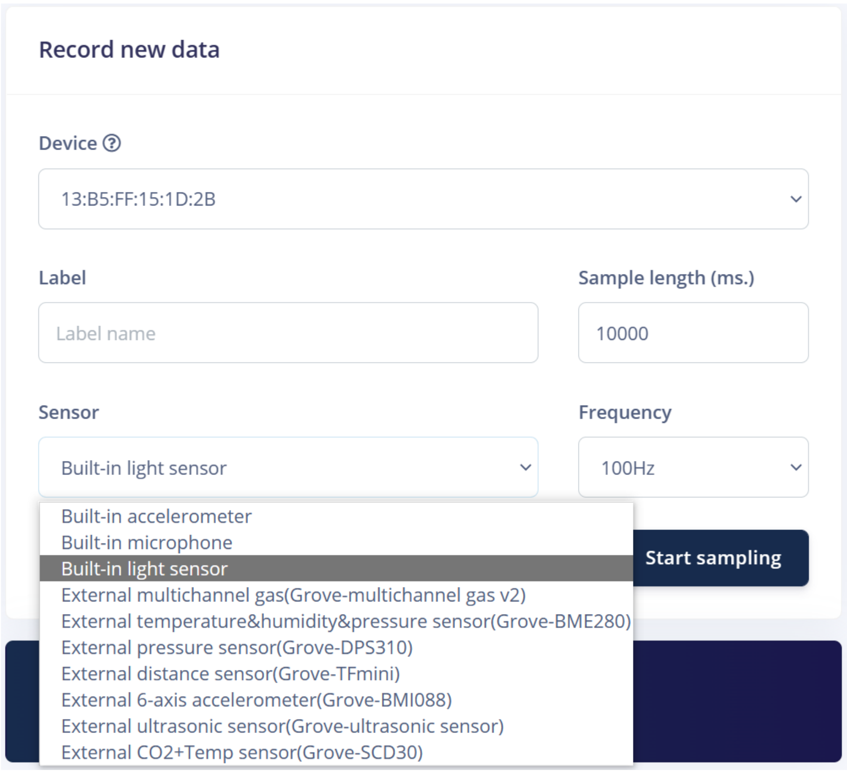
Step1: Connect Wio Terminal with Edge Impulse

Step2: Know what we are going to do

We are going to train and deploy a simple neural network for classifying rock, Vulcan gestures with just a single light sensor. So we need to select the sensor we are going to use

-- the built-in light sensor, then know what kind of data we are going to sample --gestures of Rock & Vulcan.

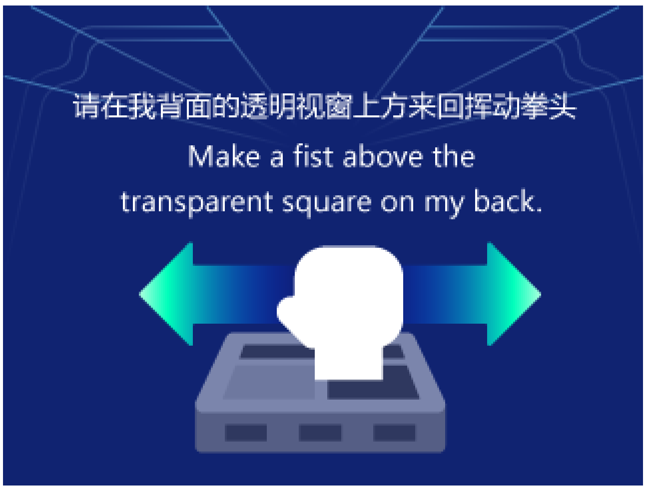
Select the sensor we are going to use -- Built-in light sensor



This indicates that we want to record data for 10 seconds (Sample length 10000ms), use a built-in light sensor and set the frequency to 100Hz.

Know the data we are going to sample -- Rock & Vulcan

Rock

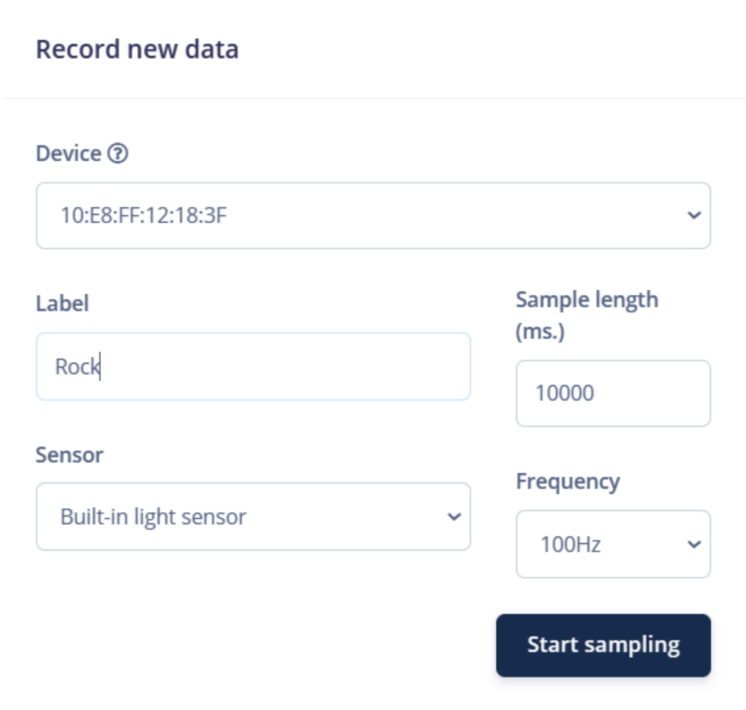


Vulcan

Environment

Sample the data of the Environment around you.

Step3: Sample



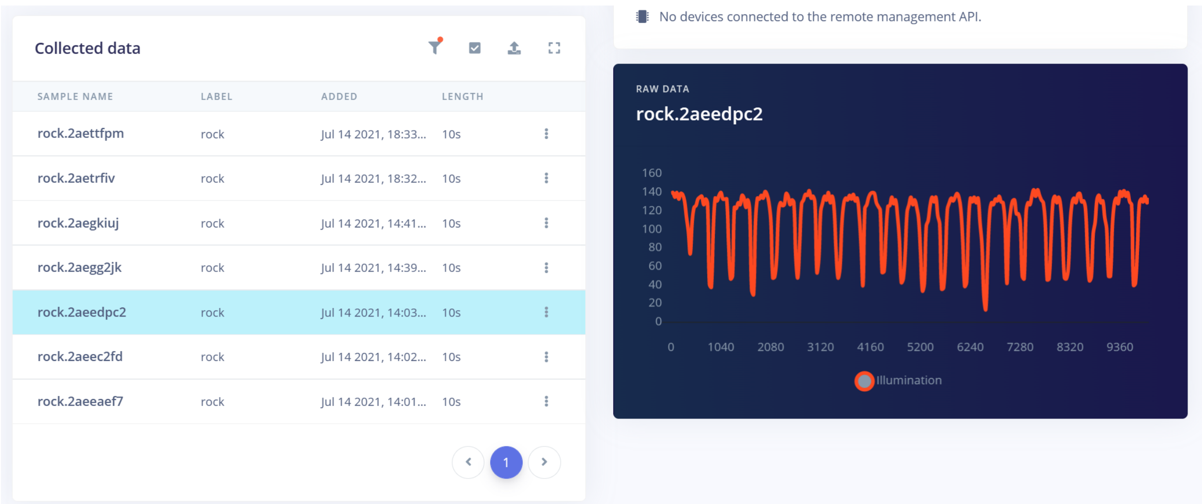
Enter the label(label the recorded data as “Rock” here, we can later edit), click "Start Sampling", then do the gesture "Rock" shown in step 2 **in a continuous motion**. In about twelve seconds the device should complete sampling and upload the file back to Edge Impulse.

Attention:

1. The light sensor location.
2. 

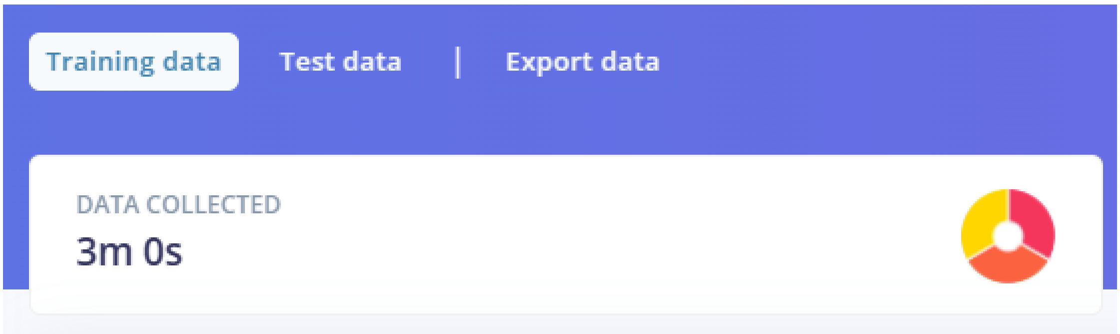
We see a new line appear under 'Collected data' in the studio.

We will be able to preview the data collected after the sample collection is finished. Make sure that the data is valid before proceeding to collect the next sample.



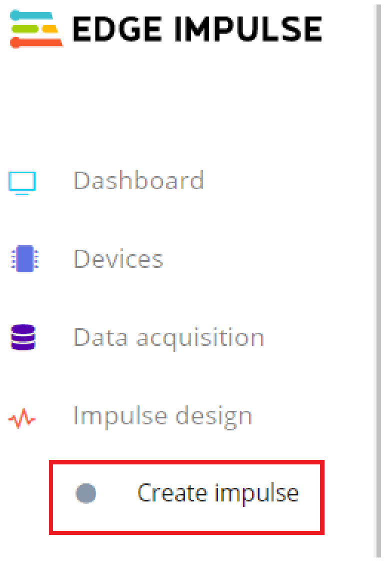
Do the same thing to sample “Vulcan” and “Environment”.

Now, we have recorded around 1 minute of data per class:

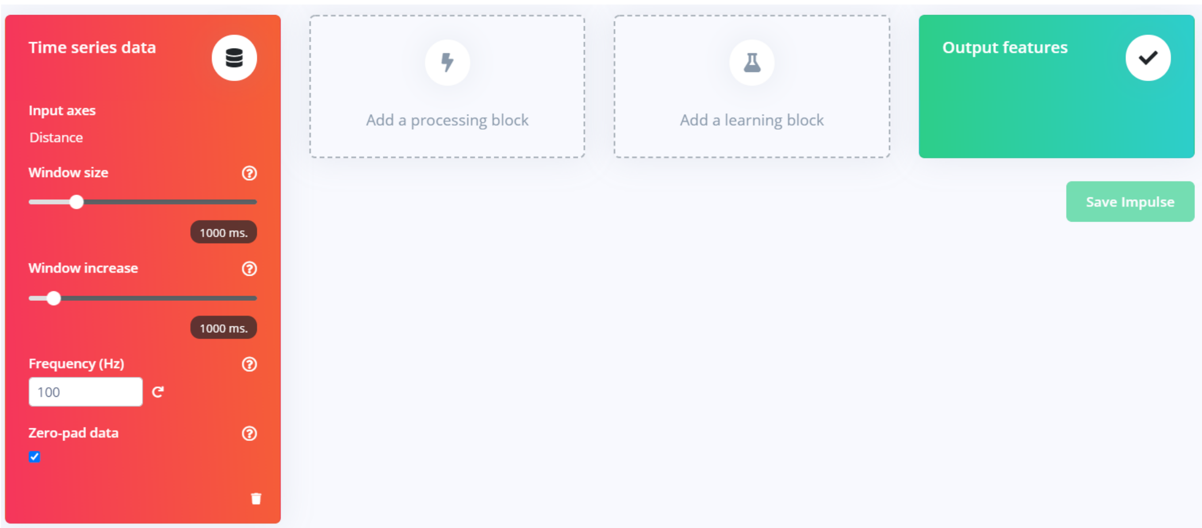


Impulse Design

After you collected the samples it is time to design an "impulse".

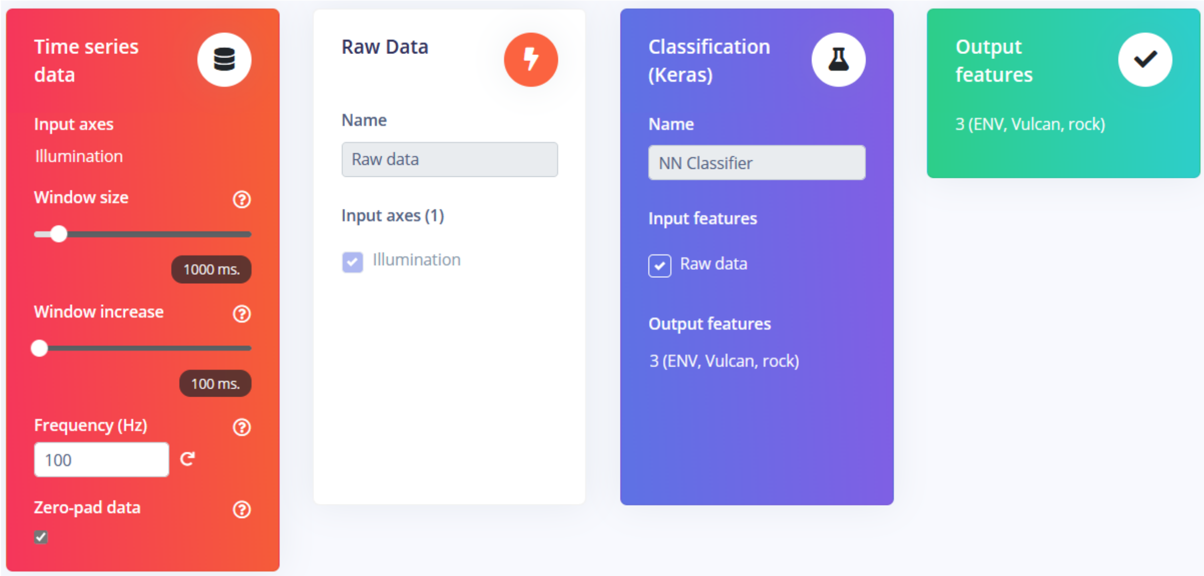


The impulse here is the word Edge Impulse used to denote data processing – training pipeline.



An impulse takes the raw data, slices it up in smaller windows, uses signal processing blocks to extract features, and then uses a learning block to classify new data.

Set as follows in this project.

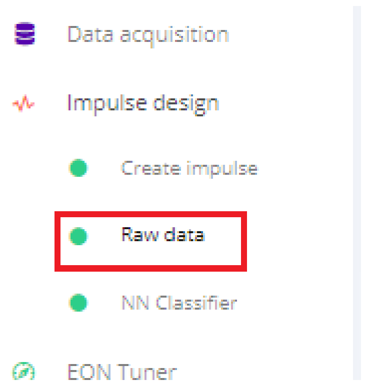


These settings mean that each time an inference is performed we're going to take sensor measurements for 1000 ms. - how many measurements your device is going to take depends on the frequency. During data collection, we set the sampling frequency to 100 Hz or 100 times per 1 second. So, to sum it up, our device is going to gather 100 data samples within 1000 ms. time window and then take these values, preprocess them and feed them to the neural network to get inference results. Of course, we use the same window size during the training.

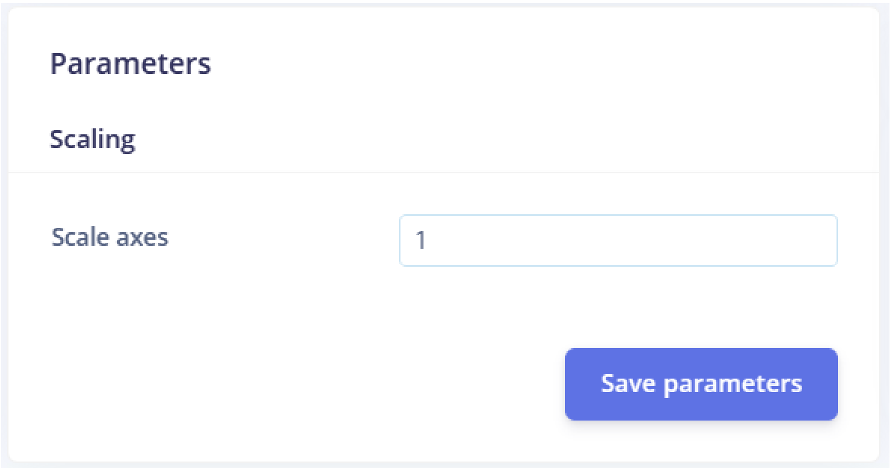
In all, set the window size to 1000 (you can click on the 1000 ms. text to enter an exact value), the window increases to 100, add the 'Raw data' and 'Classification(Keras)' blocks. Then click Save impulse.

Feature Extraction--RAW Data

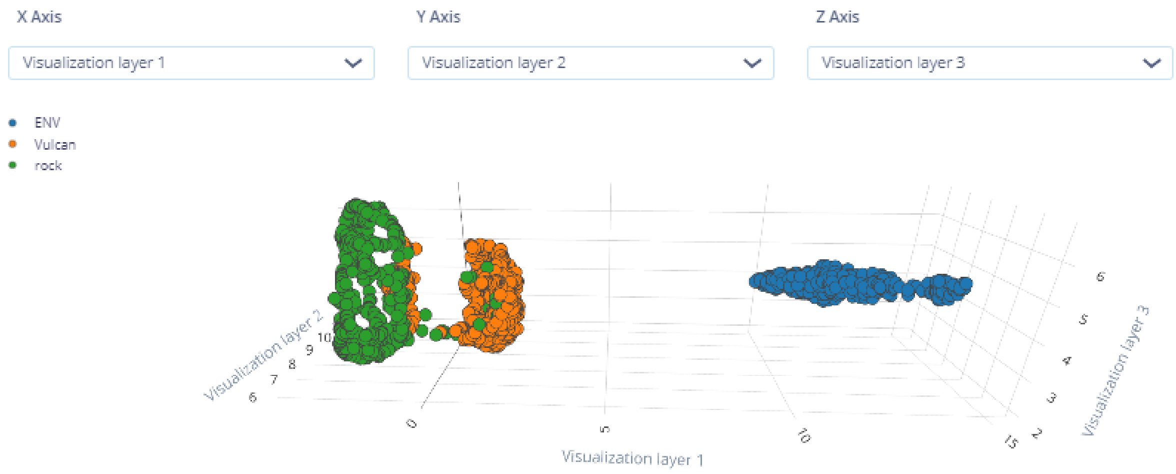
To configure your signal processing block, click Raw data in the menu on the left.



we use the default parameter and click “Save parameters” then click “Generate features” to start the process.

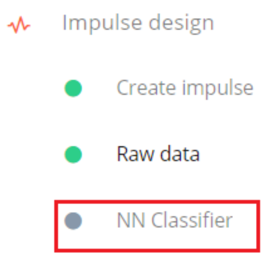


Afterwards the 'Feature explorer' will load. This is a plot of all the extracted features against all the generated windows. You can use this graph to compare your complete data set. A good rule of thumb is that if you can visually separate the data on a number of axes, then the machine learning model will be able to do so as well.

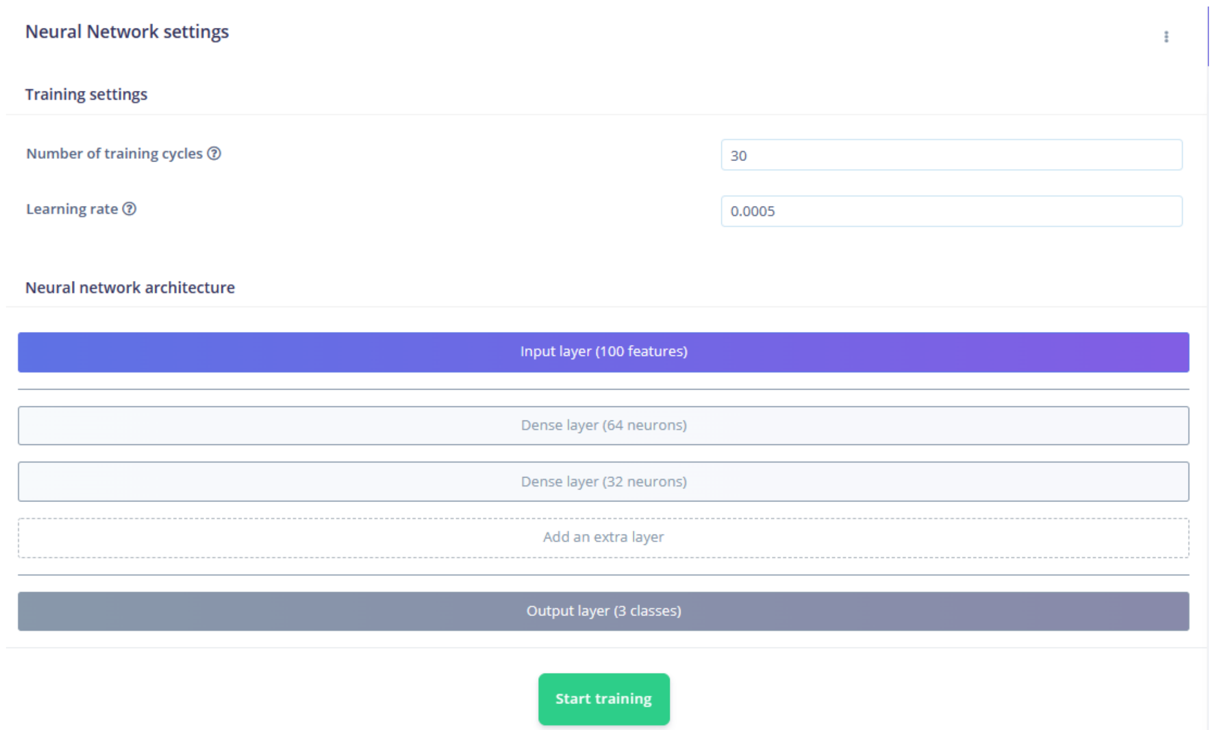


Model Training--Network: MLP

To configure our learning block, click “NN Classifier” in the menu on the left.

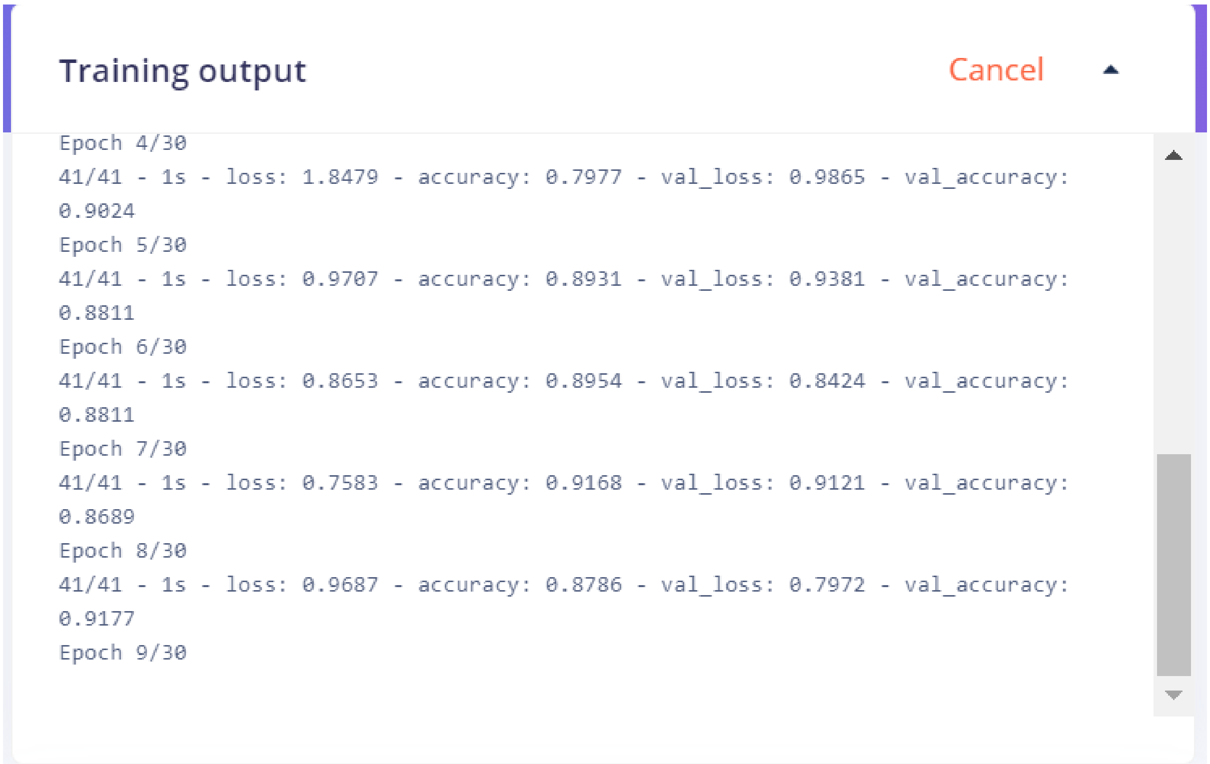


We create a simple neural network – MLP（Multilayer perceptron）here.



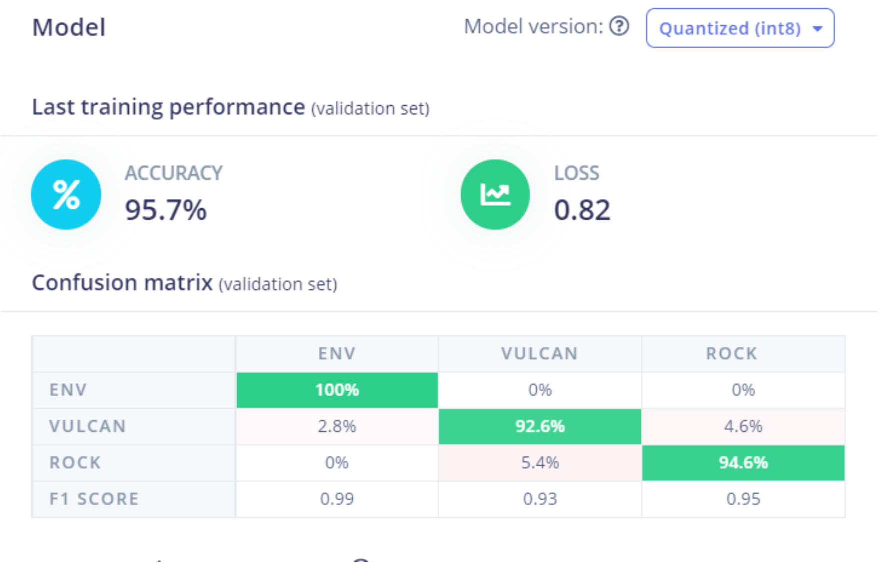
Click on“Start training”.

By observing the “Log”, you can deduce the model performance.



The number of Training Cycles is also called an epoch. It specifies the number of times that the entire training dataset passes through the training process of the algorithm. With each epoch, the internal model parameters will update, which may help the model perform better.

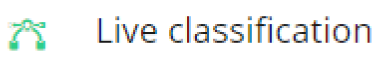
As the training log shows, “Epoch: 8/30” indicates the total number of training rounds is 30 while 8 rounds have been trained. Training accuracy has been achieved 0.8786.



	ENV	VULCAN	ROCK
ENV	100%	0%	0%
VULCAN	2.8%	92.6%	4.6%
ROCK	0%	5.4%	94.6%
F1 SCORE	0.99	0.93	0.95

Now, the Model training is complete.

After we have our model and are satisfied with its accuracy in training, we can test it on new data in the Live classification tab.

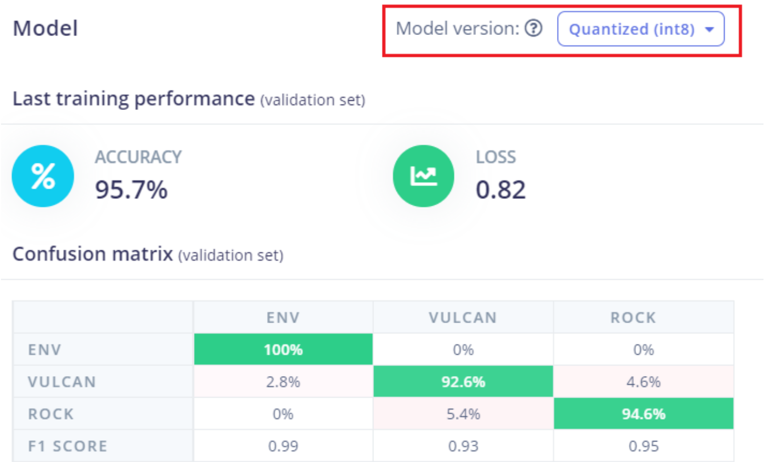


If the live classification gets poor performance, we need to analyze the reason and retrain our model. We will talk about it in future projects.

Model Optimization

There are various ways of optimizing machine learning models: compression, pruning and quantization. Compression is the process of reducing the size of a machine learning model. Pruning is the process of removing the weights of unimportant neurons from a machine learning model.

Quantization is the process of converting floating point numbers to integers.



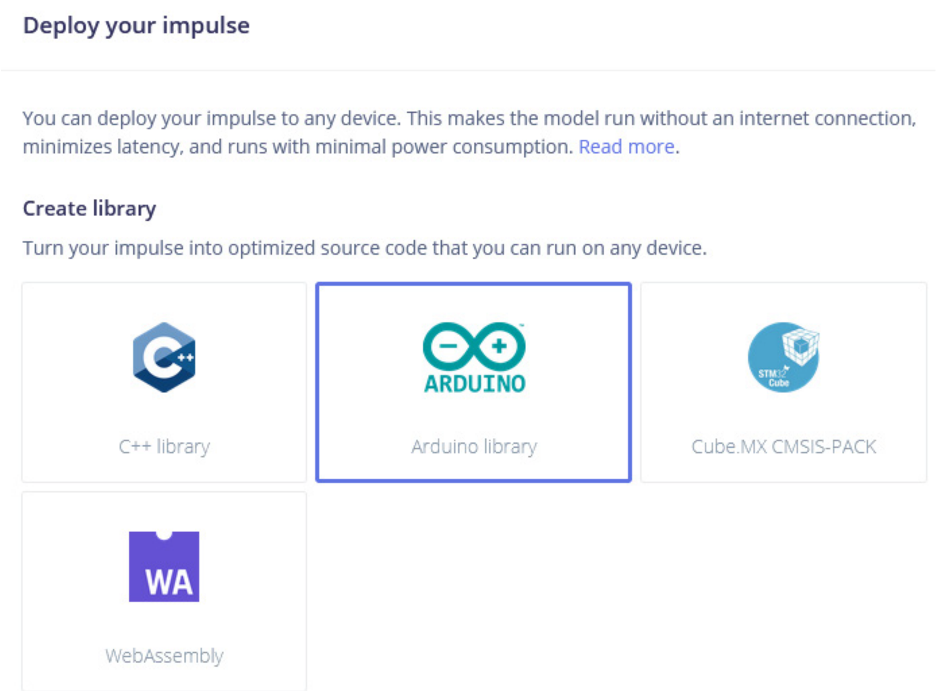
	ENV	VULCAN	ROCK
ENV	100%	0%	0%
VULCAN	2.8%	92.6%	4.6%
ROCK	0%	5.4%	94.6%
F1 SCORE	0.99	0.93	0.95

This is done to save space and time. These optimisations not only make the model run faster but also help to reduce the memory consumption requirements of the system.

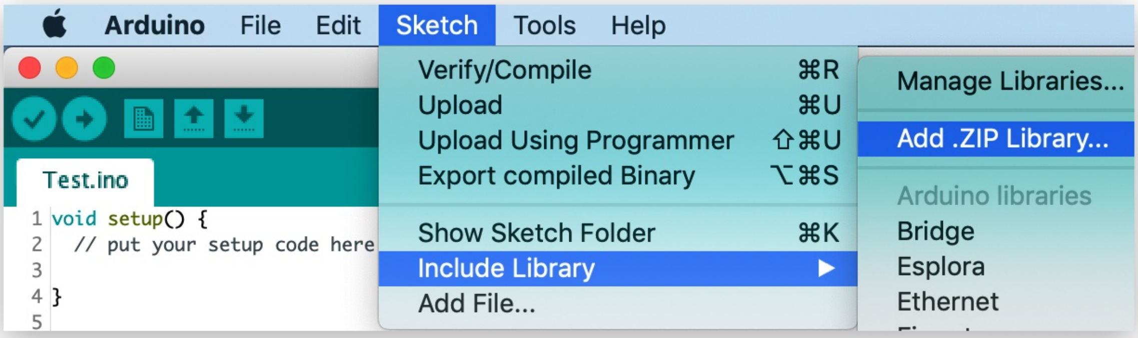
Model Deployment

The next step is deployment on the device.

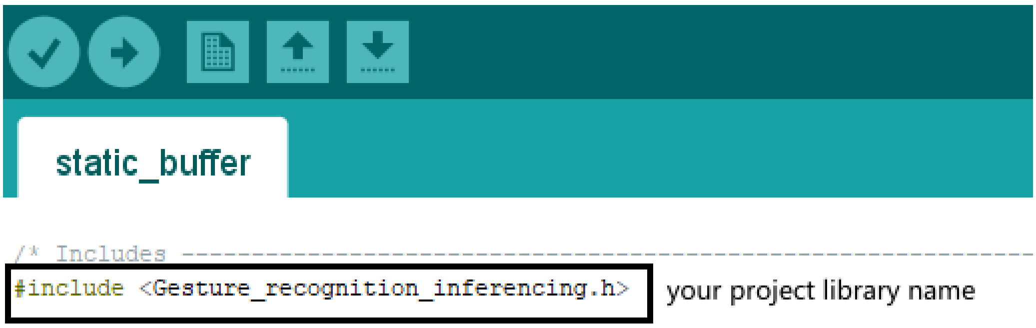
Step 1: After clicking on the Deployment tab, choose Arduino library and download it.



Step 2: Now, the library can be installed to the Arduino IDE. Open the Arduino IDE, click sketch -> Include Library -> Add .ZIP Library, and choose the file that you have just downloaded.



Step 3: Open Examples -> name of your project -> static buffer.

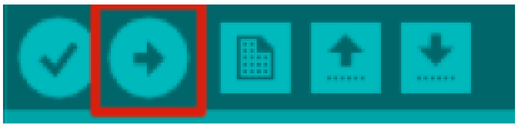


Step 4: Copy the following **Example Code** to replace the original example code:

```
#include <project_71231_inferencing.h> //replace with your project library name

ei_impulse_result_classification_t
currentClassification[EI_CLASSIFIER_LABEL_COUNT];
const char* maxConfidenceLabel;

void runClassifier()
{
  float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };
  uint8_t axis_num = 1;
  for (size_t ix = 0; ix < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; ix += axis_num) {
    uint64_t next_tick = micros() + (EI_CLASSIFIER_INTERVAL_MS * 1000);
    buffer[ix + 0] = analogRead(WIO_LIGHT);
    delayMicroseconds(next_tick - micros());
  }

  signal_t signal;
```

```
  int err = numpy:: signal_from_buffer(buffer,
EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
  ei_impulse_result_t result = { 0 };

  err = run_classifier(&signal, &result, false);
  float maxValue = 0;
  for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
    ei_impulse_result_classification_t classification_t =
result.classification[ix];
    ei_printf("    %s: %.5f\n", classification_t.label, classification_t.value);
    float value = classification_t.value;
    if (value > maxValue) {
      maxValue = value;
      maxConfidenceLabel = classification_t.label;
    }
    currentClassification[ix] = classification_t;
  }
}

void setup(){
  Serial.begin(9600);
}

void loop(){
  runClassifier();
  Serial.println(maxConfidenceLabel);
}
```

Step 5: Upload the code.



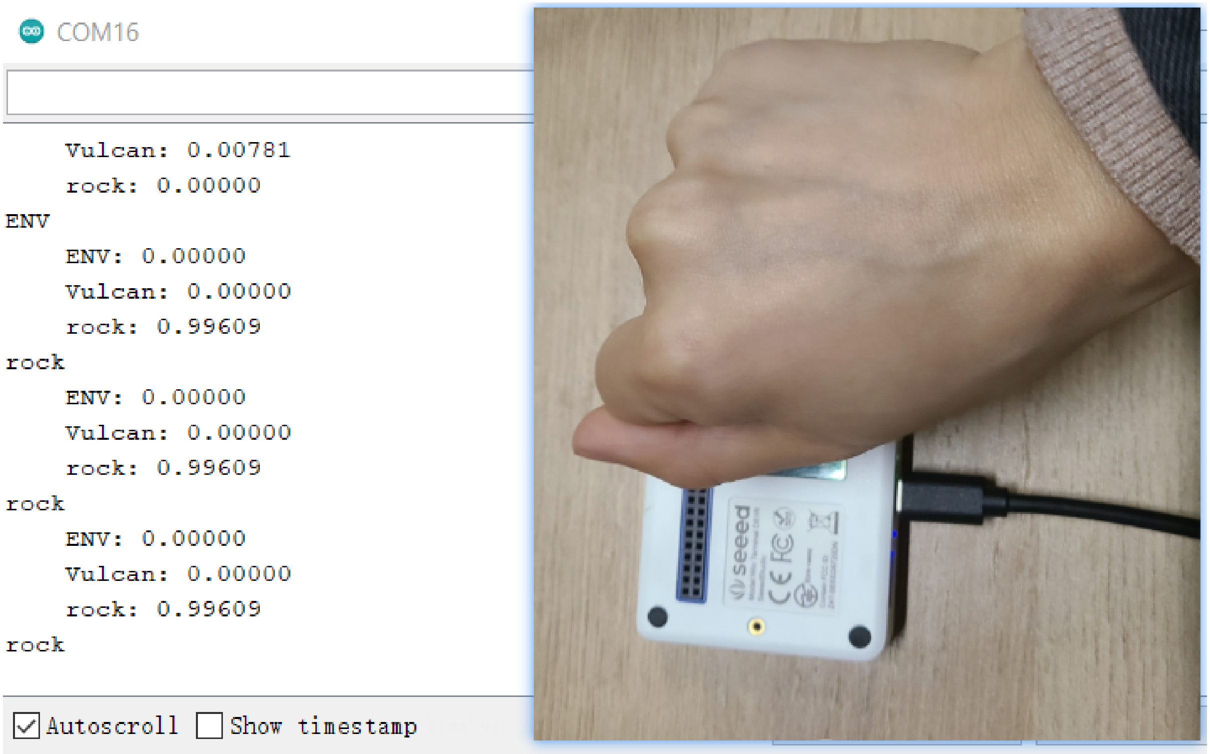
It takes about 5 mins to upload. If the upload is successful, the message "Done uploading." will appear in the status bar.

Step 6: Open the Serial monitor. Move your hand while performing a gesture and see the probability result printed out on the Serial monitor.

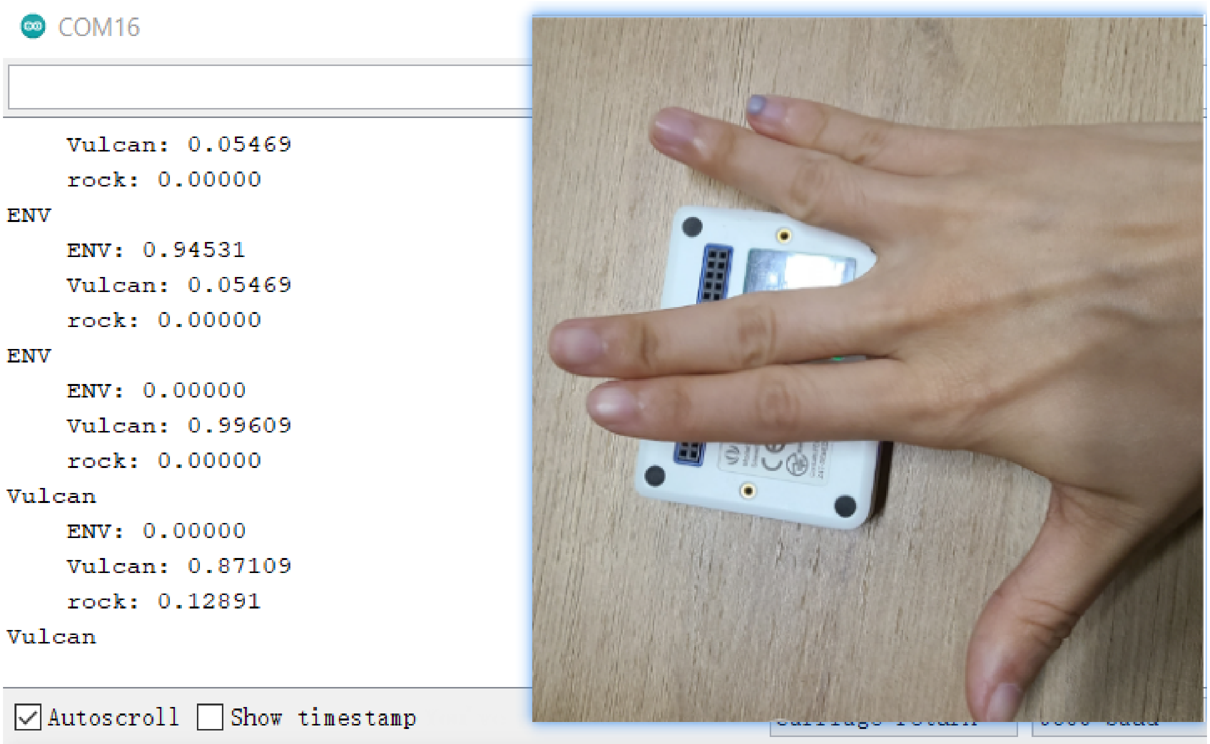
Environment


COM16
Vulcan: 0.02344
rock: 0.00000
ENV
ENV: 0.87109
Vulcan: 0.12891
rock: 0.00000
ENV
ENV: 0.87109
Vulcan: 0.12891
rock: 0.00000
ENV
ENV: 0.99219
Vulcan: 0.00781
rock: 0.00000
ENV
Autoscroll
Show timestamp
seeed

Rock:


COM16
Vulcan: 0.00781
rock: 0.00000
ENV
ENV: 0.00000
Vulcan: 0.00000
rock: 0.99609
rock
ENV: 0.00000
Vulcan: 0.00000
rock: 0.99609
rock
ENV: 0.00000
Vulcan: 0.00000
rock: 0.99609
rock
Autoscroll
Show timestamp
seeed

Vulcan:



Now, the model may perform well with recognizing your gestures but is poor at dealing with other people's “Rock”& “Vulcan”.

You can optimize your model as follows:

When collecting samples it is important to provide diversity for the model to be able to generalize better, for example, have samples with different directions, speeds and distances from the sensor. In general, the network only can learn from data present in the dataset – so if the only samples you have are gestures being moved from left to right above the sensor, you shouldn’t expect the trained model to be able to recognize gestures being moved right to left or up and down.

Collect more data samples in different light conditions.

While it was just a proof of concept demonstration, it really shows TinyML is up to something big. You probably knew it is possible to recognize gestures with a camera sensor, even if the image is down-scaled a lot. But recognizing gestures with just 1 pixel is an entirely different level!

Congratulations, we have done our first project!

Reference

Edge Impulse Public project:

https://studio.edgeimpulse.com/public/76967/latest

Practice 2. Motion Recognition using built-in accelerometer (Flip, Wave, Idle)

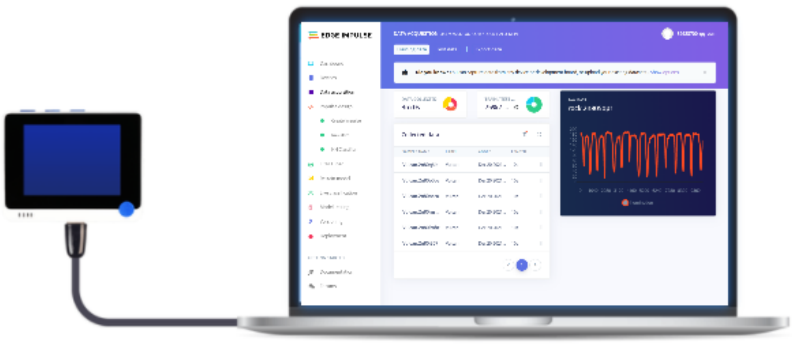
Project Overview

In this lesson, we'll take on a similar task, motion recognition, but will use a different sensor for that - a 3-axis accelerometer. This is a hard task to solve using rule-based programming, as people don't perform gestures in the exact same way every time. But machine learning can handle these variations with ease.

Material Preparation

Hardware requirements: Wio Terminal

Connection method:



About sensor: accelerometers

As you might guess from the name, accelerometers are devices that measure the acceleration of a body. It is defined as the rate of change of the velocity of an object. Accelerometers are capable of measuring acceleration either in meters per second squared (m/s2) or in G-forces (g). A single G-force on Earth is equivalent to 9.8 m/s2. There are different kinds of accelerometers. The earliest accelerometers were based on mechanical architecture.

The first accelerometer was called the Atwood machine. It was invented by an English physicist George Atwood.

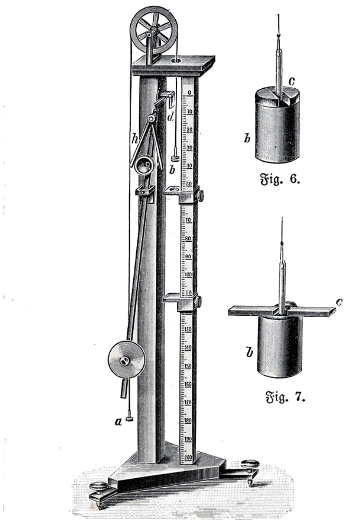
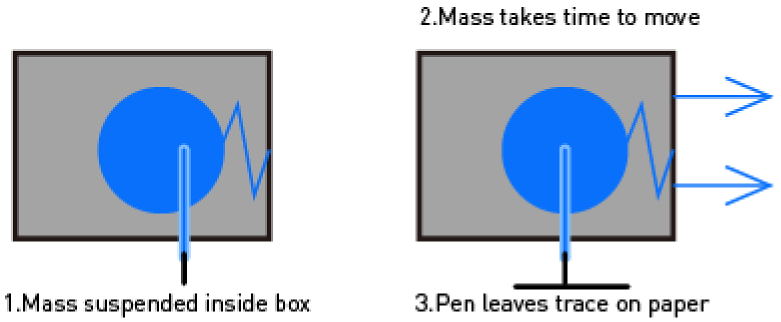
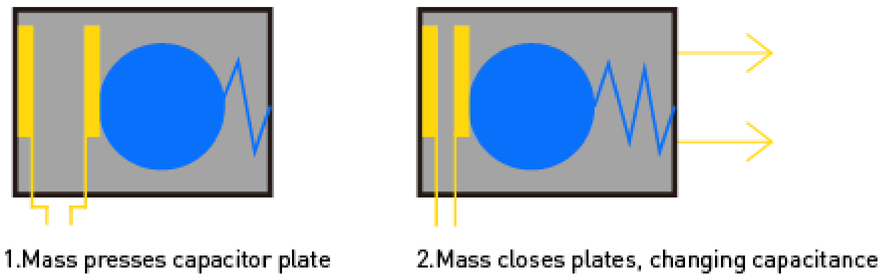


Fig. 5.
Atwoodsche Fallmaschine

The accelerometers commonly used in mobile phones are MEMS (Microelectromechanical) accelerometers.



The module used in Wio Terminal is called 3-Axis Digital Accelerometer (LIS3DHTR). Generally, the internal structure of accelerometers consists of Capacitive Plates. Some types of accelerometers use fixed capacitive plates, while some of them have the plates attached to minuscule springs that move internally depending upon the acceleration forces acting on the sensor.



Machine Learning Lifecycle

Please Open: https://www.edgeimpulse.com/

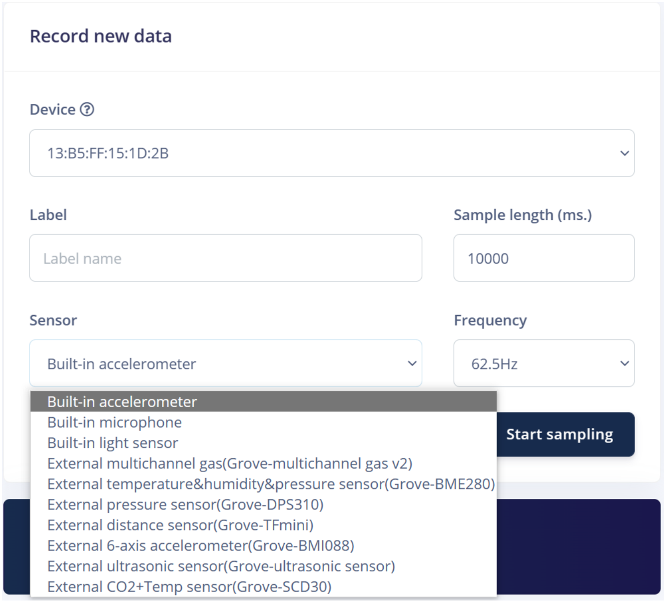
Data Collection

Step1: Connect Wio Terminal with Edge Impulse

Step2: Know what we are going to do

We are going to train and deploy a simple neural network for classifying Flip, Wave, Idle motion with just an accelerometer. So we need to select the sensor we are going to use -- a built-in accelerometer, then know what kind of data we are going to sample --the motion of “Flip”, “Wave”, “Idle”.

Select the sensor we are going to use -- Built-in accelerometer



This indicates that we want to record data for 10 seconds (Sample length 10000ms), use the built-in accelerometer and frequency 62.5Hz.

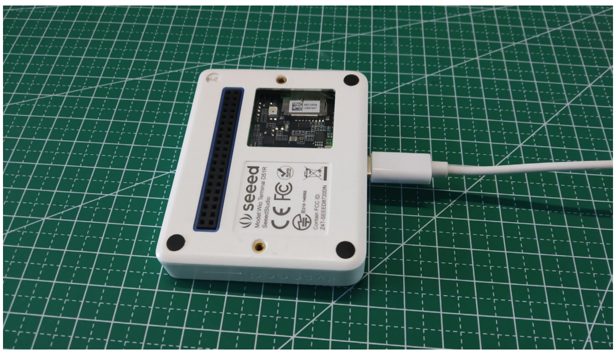
Know the data we are going to sample -- Flip, Wave, Idle

• Idle - just sitting on your desk while you're working.

• Flip - turn device, similar to how you would turn a valve

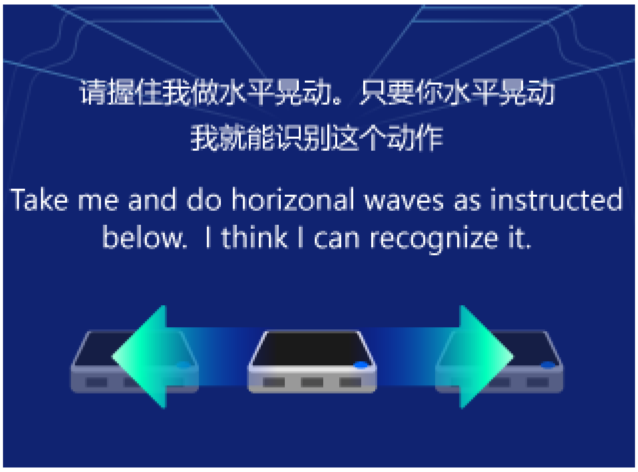
• Wave - waving the device from left to right.

Idle

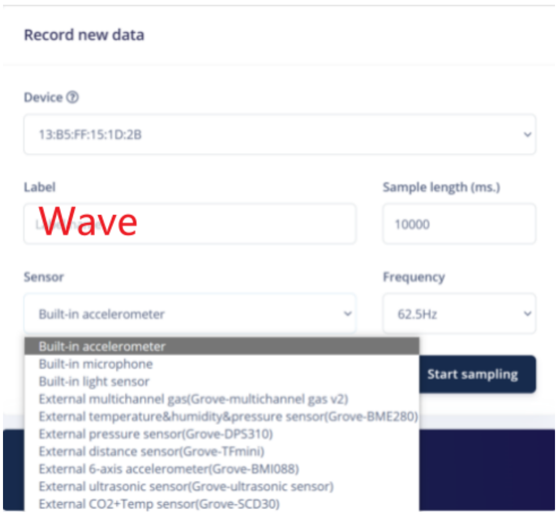


Flip

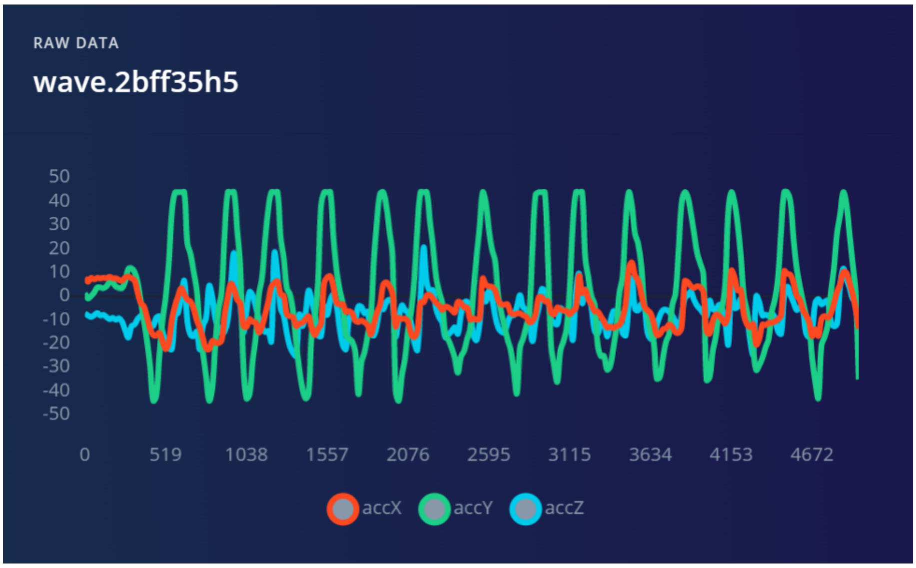
wave



Step3: Sample



Enter the label, click "Start Sampling", then do the gesture “Wave” shown in step 2 **in a continuous motion.** In about twelve seconds the device should complete sampling and upload the file back to Edge Impulse. You see a new line appear under 'Collected data' in the studio. When you click it you now see the raw data graphed out. As the accelerometer on the development board has three axes you'll notice three different lines, one for each axis.

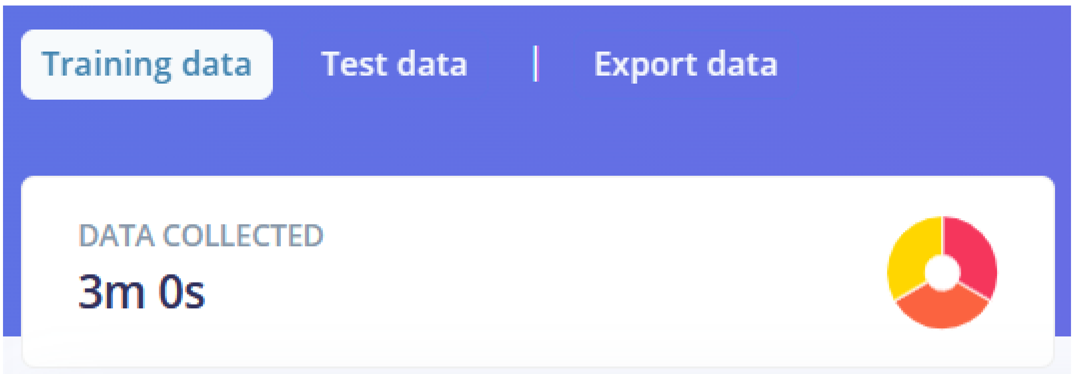


Note

Make sure to perform variations on the motions. E.g. do both slow and fast movements and vary the orientation of the board. You'll never know how your user will use the device.

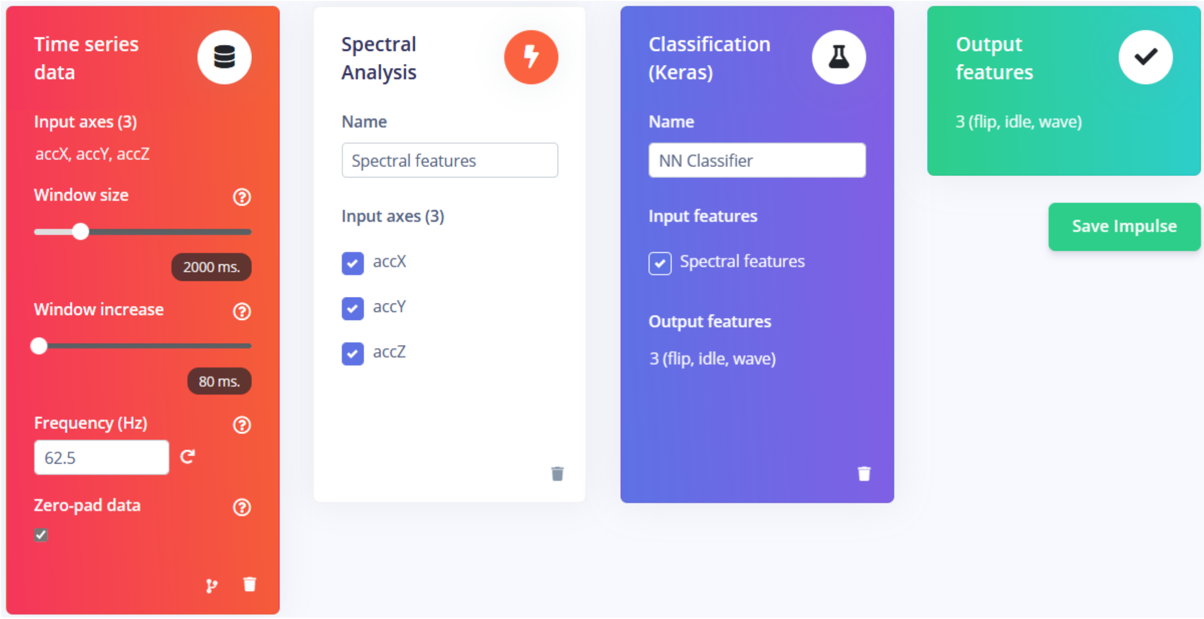
Gather data from 2 other people, except for yourself.

Now, we have recorded around 1min of data per class:



Impulse Design

With the training set in place, we can design an impulse.



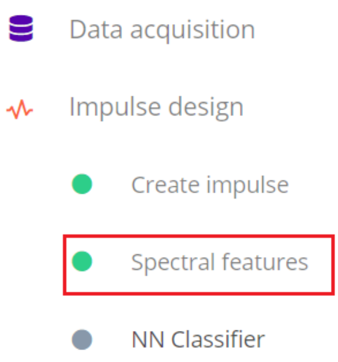
Signal processing blocks always return the same values for the same input and are used to make raw data easier to process, while learning blocks learn from past experiences. For this tutorial, we'll use the 'Spectral analysis' signal processing block. This block applies a filter, performs spectral analysis on the signal, and extracts frequency and spectral power data.

Then we'll use a 'Neural Network' learning block, that takes these spectral features and learns to distinguish between the three (Idle, Flip and Wave) classes.

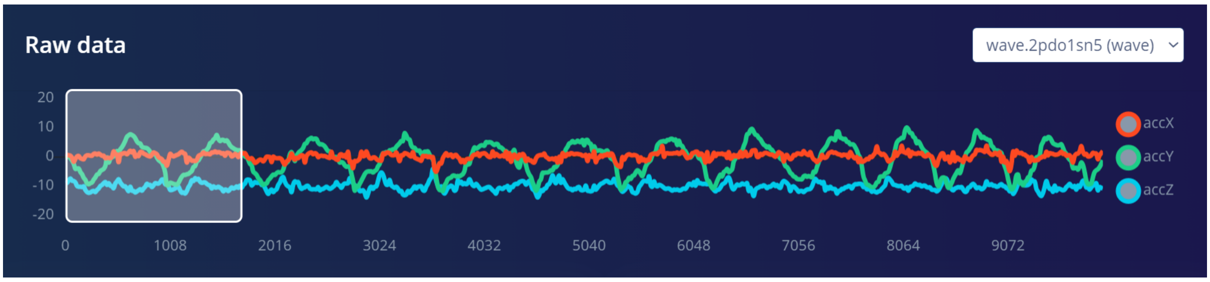
In all, set the window size to 2000 (you can click on the 2000 ms. text to enter an exact value), the window increases to 80, and adds the 'Spectral Analysis' and 'Classification (Keras)' blocks. Then click Save impulse.

Feature Extraction--Spectral Analysis

To configure your signal processing block, click Spectral features in the menu on the left.



This will show you the raw data on top of the screen (you can select other files via the drop-down menu).



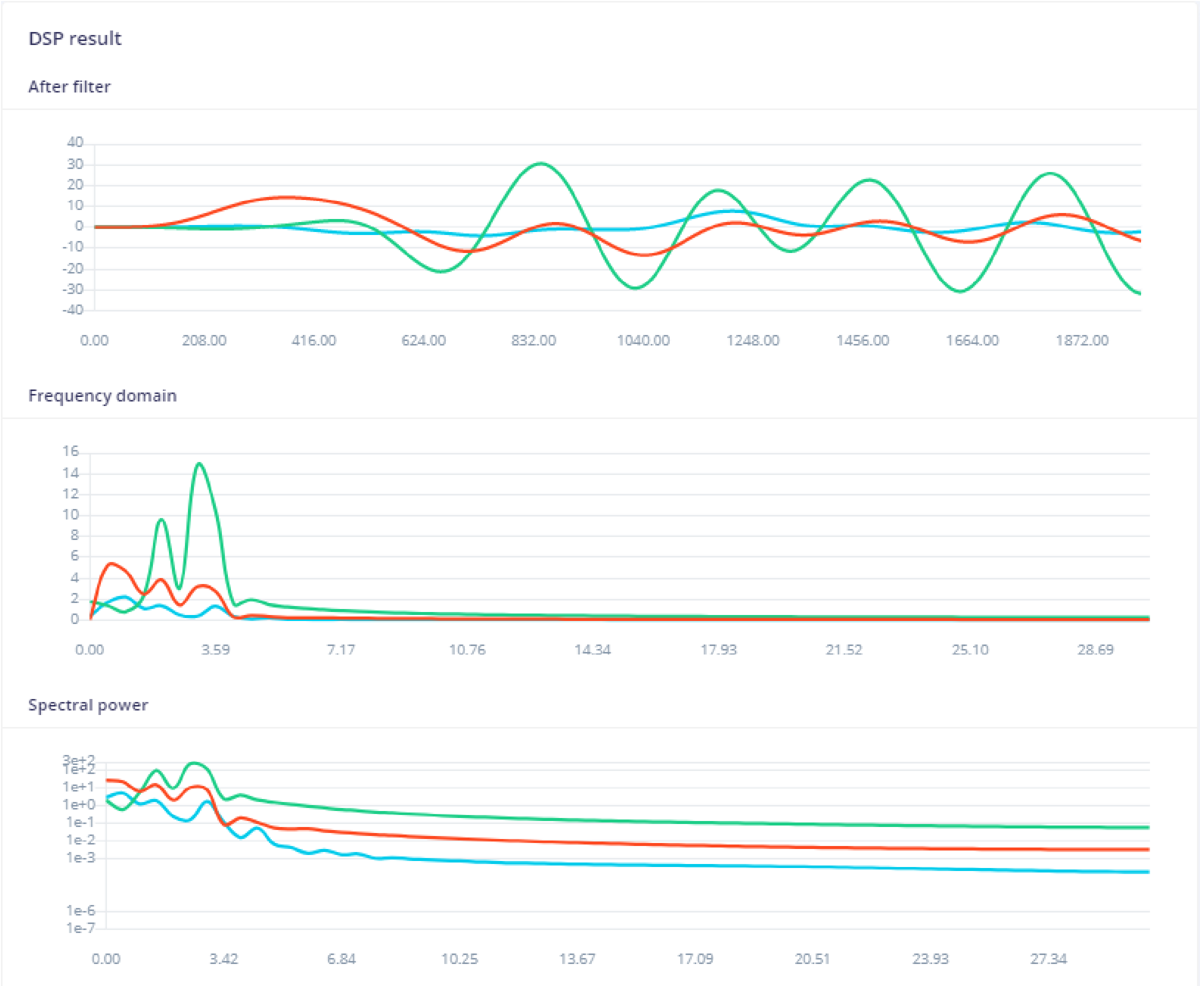
And the results of the signal processing through graphs on the right.

For the spectral features block you'll see the following graphs:

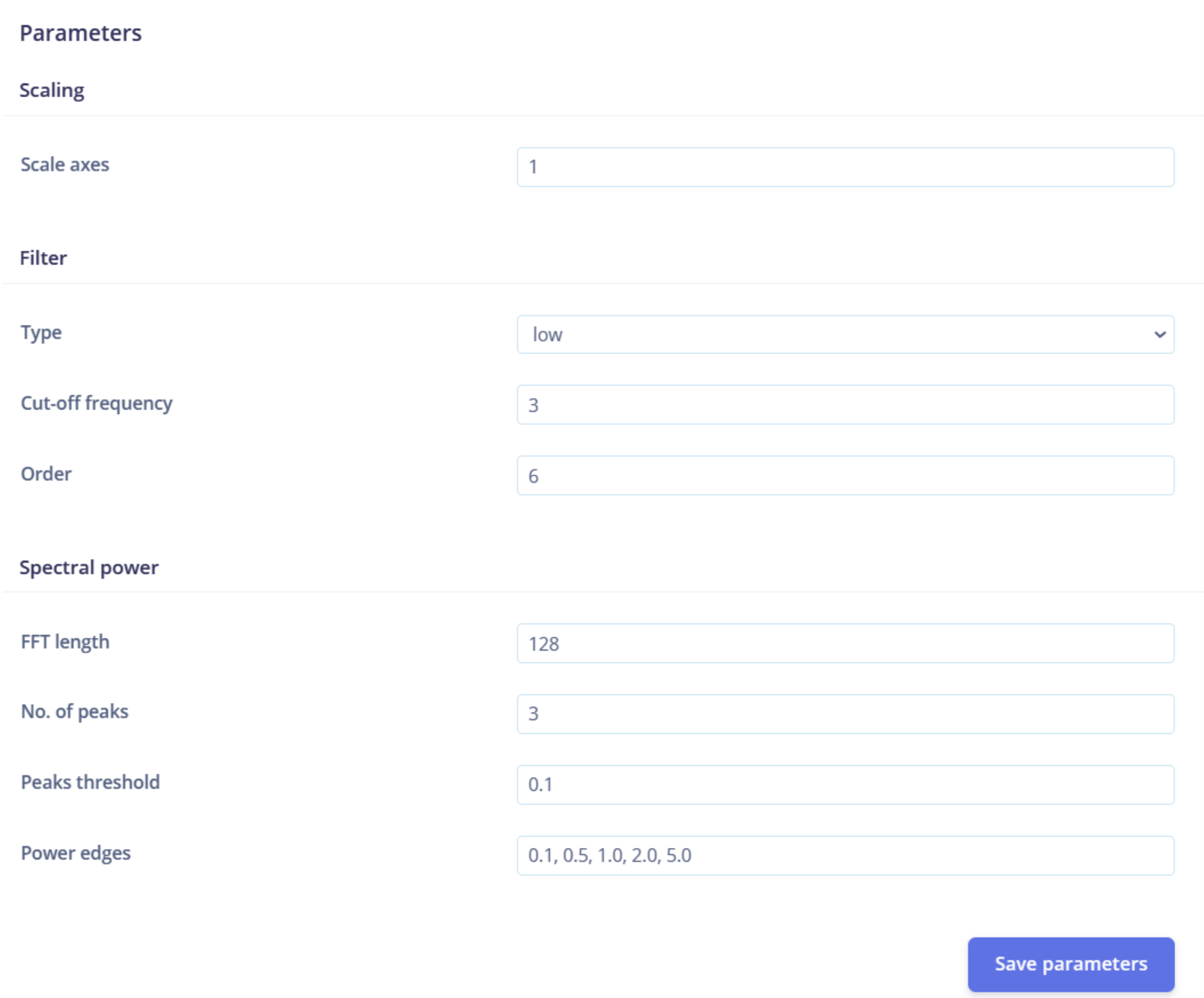
• After filter - the signal after applying a low-pass filter. This will remove noise.

• Frequency domain - the frequency at which signal is repeating (e.g. making one wave movement per second will show a peak at 1 Hz).

• Spectral power - the amount of power that went into the signal at each frequency.

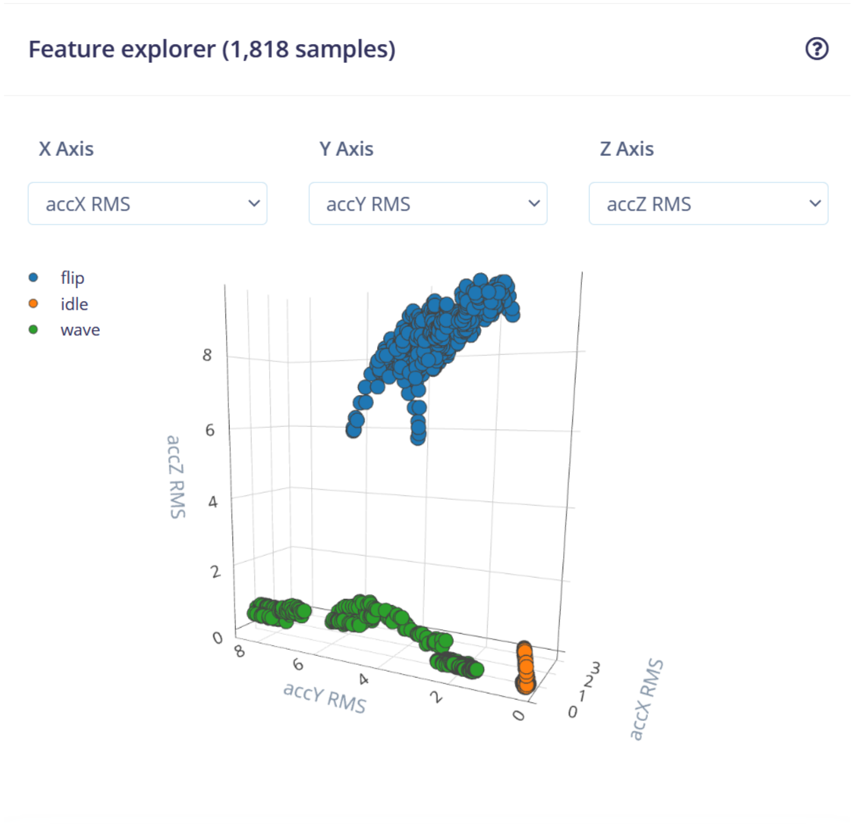


A good signal processing block will yield similar results for similar data. If you move the sliding window (on the raw data graph) around, the graphs should remain similar. Also, when you switch to another file with the same label, you should see similar graphs, even if the orientation of the device was different.



Click “Save parameters” then click “Generate features” to start the process.

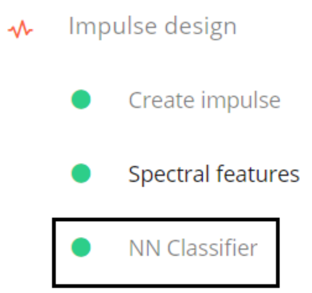
Afterwards the 'Feature explorer' will load. This is a plot of all the extracted features against all the generated windows. You can use this graph to compare your complete data set.



Model Training--Network: MLP

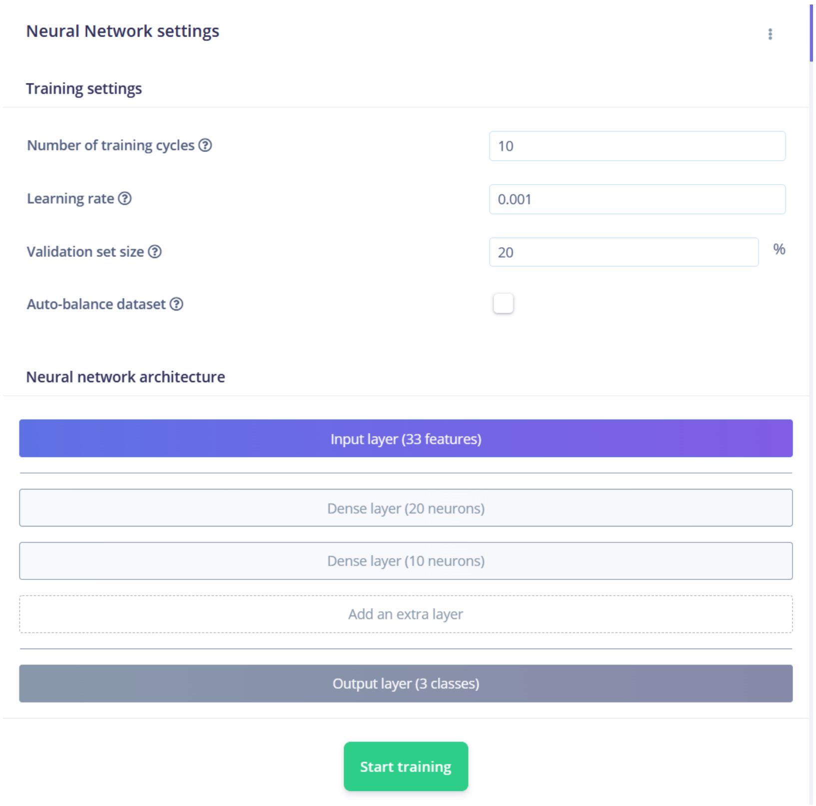
With all data processed it's time to start training a neural network.

To configure our learning block, click “NN Classifier” in the menu on the left.



Neural networks are a set of algorithms, modeled loosely after the human brain, that is designed to recognize patterns. The network that we're training here will take the signal processing data as an input, and try to map this to one of the three classes.

We create a simple neural network – MLP（Multilayer perceptron）here.



Now, the Model training is complete.

Model Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 99.7%

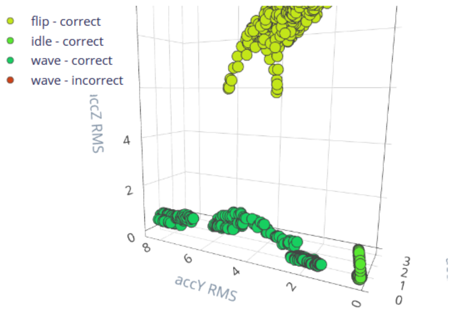
LOSS 0.02

Confusion matrix (validation set)

	FLIP	IDLE	WAVE
FLIP	100%	0%	0%
IDLE	0%	100%	0%
WAVE	0%	0.8%	99.2%
F1 SCORE	1.00	1.00	1.00

Feature explorer (full training set)

accX RMS | accY RMS | accZ RMS



On-device performance

INFERENCING TIME 1 ms.

PEAK RAM USAGE 1.7K

FLASH USAGE 19.3K

INFERENCING TIME: The time that a device takes to complete a prediction.
PEAK RAM USAGE and FLASH USAGE: Let's look at the Wio Terminal's Flash and RAM.

Powerful MCU - Microchip ATSAMD51P19

- ARM Cortex-M4F core running at **120MHz (Boost up to 200MHz)**
- **4 MB** External Flash, **192 KB** RAM

Every embedded machine learning model has constraints. The model we get this time meets these constraints.

After we have our model and are satisfied with its accuracy in training, we can test it on new data in the Live classification tab.

Live classification

If the live classification gets poor performance, we need to analyze the reason and retrain our model.

Model Optimization

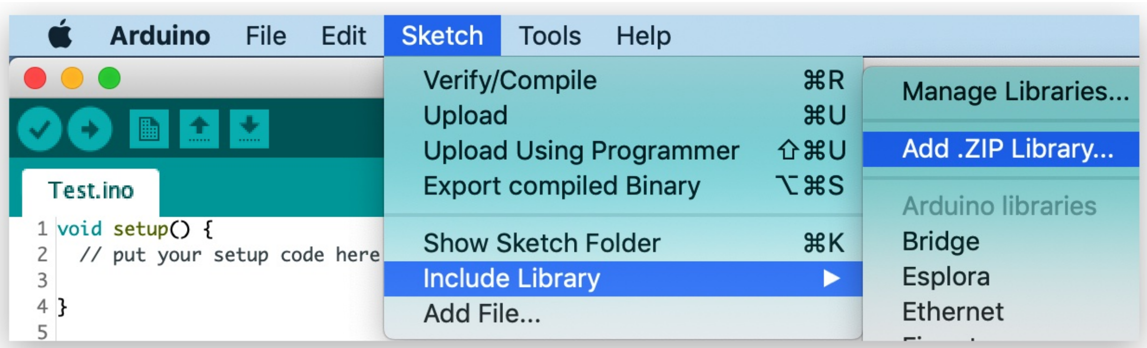
Model optimization follows Practice 1.

Model Deployment

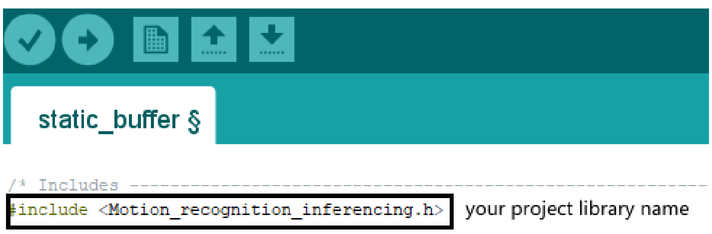
The next step is deployment on the device.

Step 1: After clicking on the Deployment tab, choose Arduino library and download it.

Step 2: Now, the library can be installed to the Arduino IDE. Open the Arduino IDE, click sketch -> Include Library -> Add .ZIP Library, and choose the file that you have just downloaded.



Step 3: Open Examples -> name of your project -> static buffer.



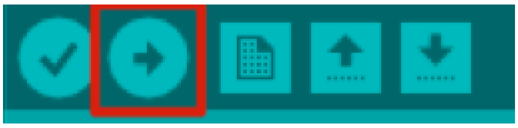
Step 4: Copy the following **Example Code** to replace the original example code:

Also, since Wio Terminal has an LCD screen, we're going to display the name of the detected class if this class confidence value is above the threshold.

```
#include"TFT_eSPI.h"
#include <project_48833_inferencing.h>//replace with your project library name
#include"LIS3DHTR.h"

TFT_eSPI tft;
LIS3DHTR<TwoWire> lis;
#define CONVERT_G_TO_MS2    9.80665f
ei_impulse_result_classification_t
currentClassification[EI_CLASSIFIER_LABEL_COUNT];
const char* maxConfidenceLabel;

void runClassifier()
{
  float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };
  for (size_t ix = 0; ix < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; ix += 3) {
    uint64_t next_tick = micros() + (EI_CLASSIFIER_INTERVAL_MS * 1000);
    lis.getAcceleration(&buffer[ix], &buffer[ix + 1], &buffer[ix + 2]);
    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());
  }
  signal_t signal;
  int err = numpy:: signal_from_buffer(buffer,
EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
  ei_impulse_result_t result = { 0 };

  err = run_classifier(&signal, &result, false);
  float maxValue = 0;
  for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
    ei_impulse_result_classification_t classification_t =
result.classification[ix];
    ei_printf("    %s: %.5f\n", classification_t.label, classification_t.value);
    float value = classification_t.value;
    if (value > maxValue) {
      maxValue = value;
      maxConfidenceLabel = classification_t.label;
    }
    currentClassification[ix] = classification_t;
  }
}

void setup(){
  tft.begin();
  lis.begin(Wire1);
  lis.setOutputDataRate(LIS3DHTR_DATARATE_100HZ);
  lis.setFullScaleRange(LIS3DHTR_RANGE_4G);
```

```
  tft.setRotation(3);
  tft.setTextSize(4);
}

void loop(){

  runClassifier();
  tft.drawString((String)maxConfidenceLabel, 120, 120);

}
```

Step 5: Upload the code.



It takes about 5 mins to upload. If the upload is successful, the message "Done uploading." will appear in the status bar.

Step 6: Wave the Wio Terminal and see the probability result printed out on the LCD screen.

Reference

Edge Impulse Public project:

https://studio.edgeimpulse.com/public/76504/latest

Practice 3.Keyword Spotting using built-in microphone. (Hello Wio)

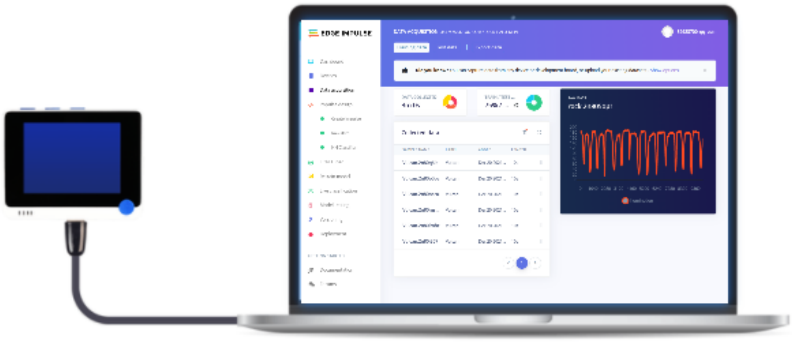
Project Overview

This model uses Wio Terminal built-in microphone to collect vocal wake words and ambient sounds to train the model. This microphone also helps to wake up the device with the wake-up word ("Hello Wio").

Material Preparation

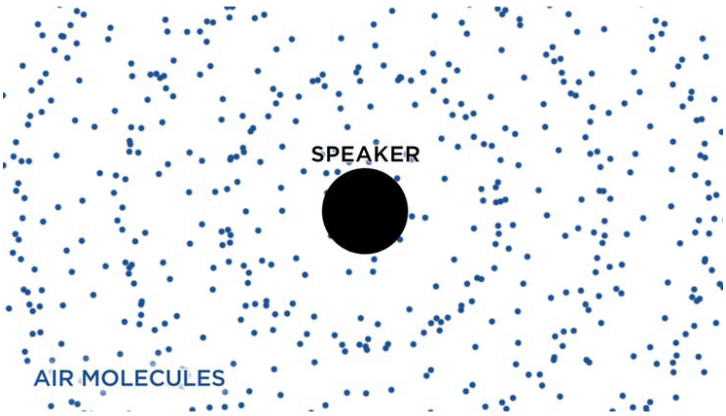
Hardware requirements: Wio Terminal

Connection method:

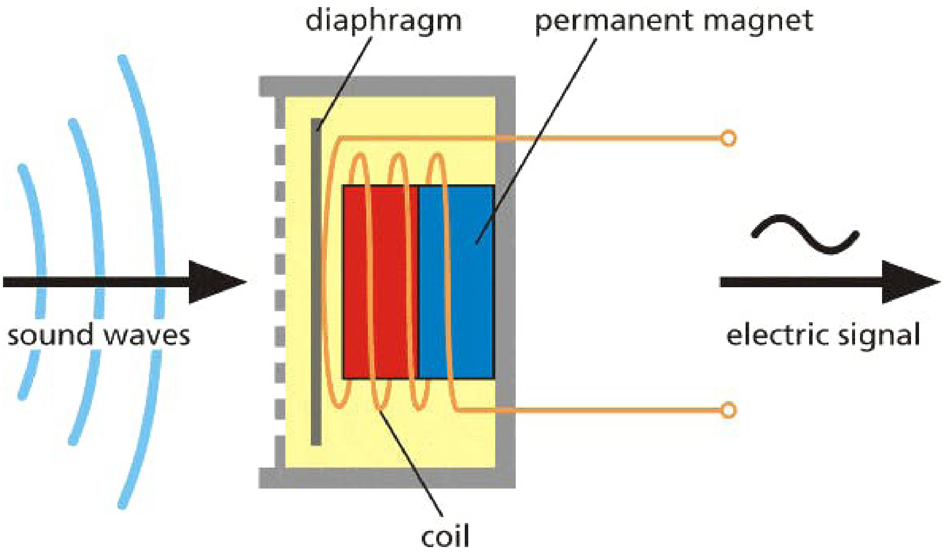


About sensor

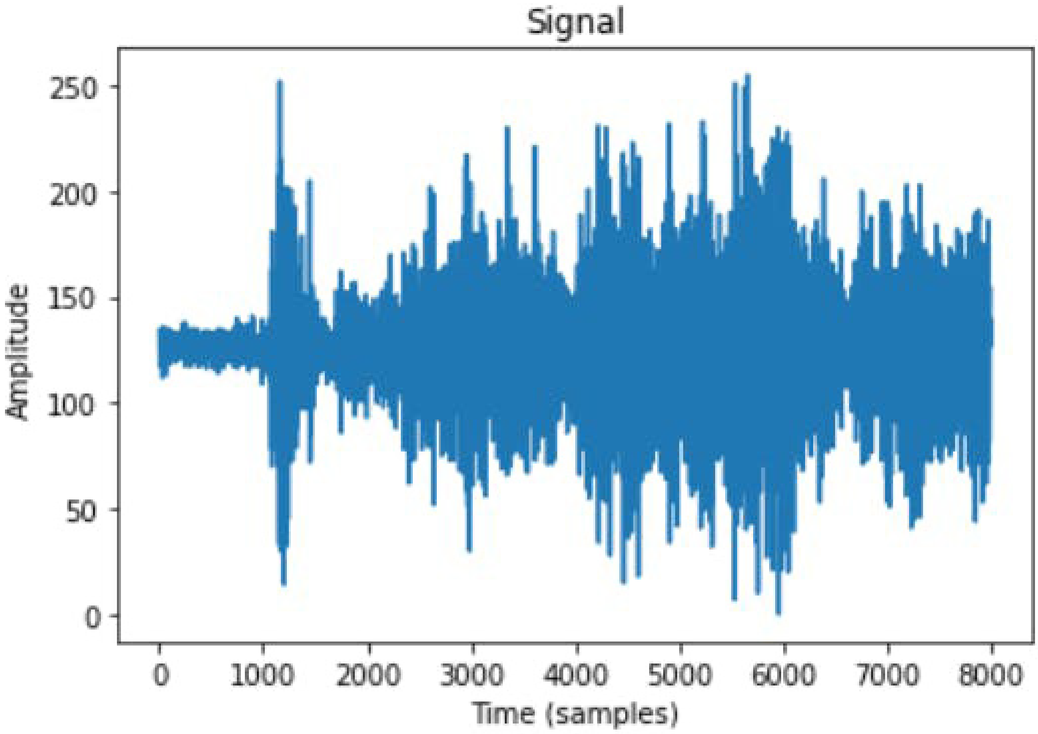
Sound is a vibration that propagates (or travels) as an acoustic wave, through a a transmission medium such as a gas, liquid or solid.



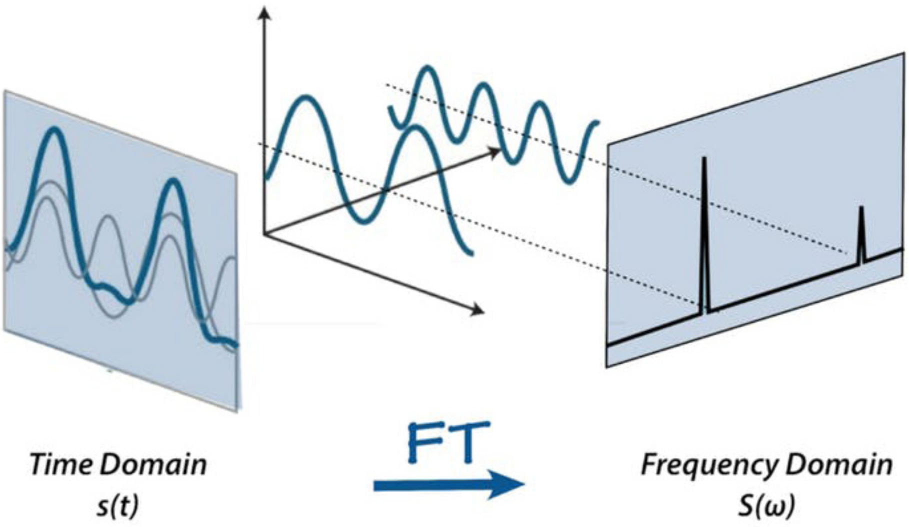
The source of sound pushes the surrounding medium molecules, they push the molecules next to them and so on and so forth. When they reach other objects it also vibrates slightly – we use that principle in the microphone. The microphone membrane gets pushed inward by the medium molecules and then back to its original position.



That generates an alternating current in the circuit, where voltage is proportional to the sound amplitude – the louder the sound, the more it pushes the membrane, thus generating higher voltage. We then read this voltage with an analogue-to-digital converter and record at equal intervals – the number of times we take measurement of sound in one second is called a sampling rate, for example, 8000 Hz sampling rate is taking measurement 8000 times per second. Sampling rate obviously matters a lot for the quality of the sound – if we sample too slow we might miss important bits and pieces. The numbers used for recording sound digitally also matter – the larger range of a number used, the more“nuances” we can preserve from the original sound. That is called audio bit depth – you might have heard terms like 8-bit sound and 16-bit sound. Well, it is exactly what it says on the tin – for 8-bit sound unsigned 8-bit integers are used, which have ranged from 0 to 255. For 16-bit sound signed 16-bit integers are used, so that’s -32768 to 32767. Alright, so in the end we have a string of numbers, with larger numbers corresponding to loud parts of the sound and we can visualize it like this - this is 1 second of gunshot sound recorded at 8000 Hz frequency in 8-bit depth (0-255).



We can't do much with this raw sound representation though – yes, we can cut and paste the parts or make it quieter or louder, but for analyzing the sound, it is, well, too raw. Here is where Fourier transform, Mel scale, spectrograms and cepstrum coefficients come in. For the purpose of this project, we'll define the Fourier transform as a mathematical transform that allows us to decompose a signal into its individual frequencies and the frequency's amplitude.



Machine Learning Lifecycle

Please Open: https://www.edgeimpulse.com/

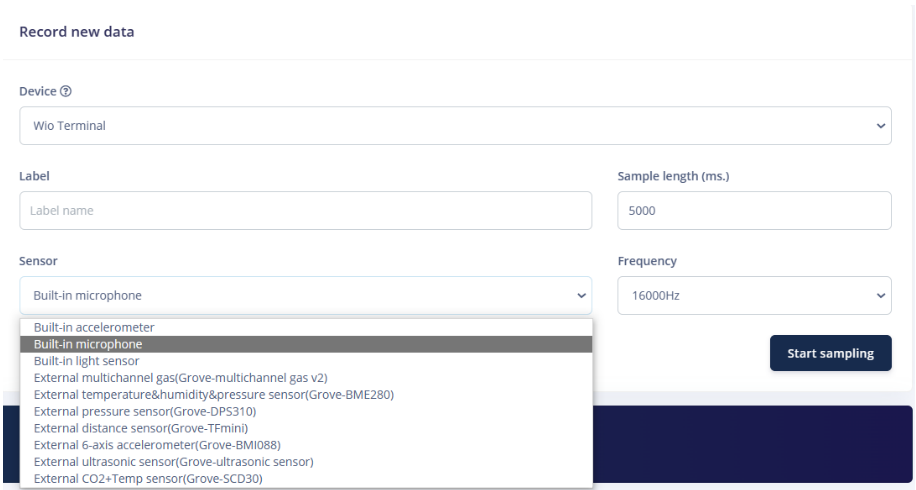
Data Collection

Step1: Connect Wio Terminal with Edge Impulse

Step2: Know what we are going to do

We are going to train and deploy a simple neural network for keyword spotting with just a built-in microphone. So we need to select the sensor we are going to use -- Built-in microphone, then know what kind of data we are going to sample --keyword.

Select the sensor we are going to use -- a Built-in microphone



This indicates that we want to record data for 5 seconds (Sample length 5000ms), use the built-in microphone and frequency 16000 Hz.

Know the data we are going to sample.

We want to build a system that recognizes keywords, so our first job is to think of a great one. It can be the name of your device, the name of your pet, etc. But keep in mind that some keywords are harder to distinguish from others, and especially keywords with only one syllable might lead to false- positives(like 'Hi'). This is the reason that Apple, Google and Amazon all use at least three-syllable keywords ('Hey Siri', 'OK, Google', 'Alexa').

So we choose "Hello Wio", and say hello to our Wio Terminal.

Hello Wio

In addition to our keyword, we'll also need audio that is not our keyword. Like background noise, and humans saying other words.

Background

Acquire background sound

Record the ambient background sound

Unknown

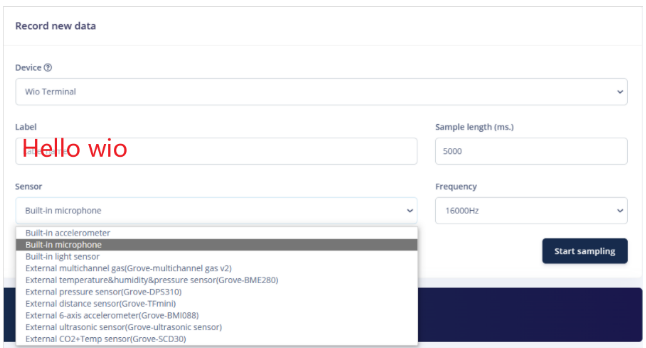
other words

Say some common words different from the wake-up wor

This is required because a machine learning model has no idea about right and wrong, but only learns from the data we feed into it.

So we should tell the machine learning model; when you hear this, this is background, when you hear that, that is unknown words. and only when you hear "hello Wio "that is Hello Wio.

Step3: Sample

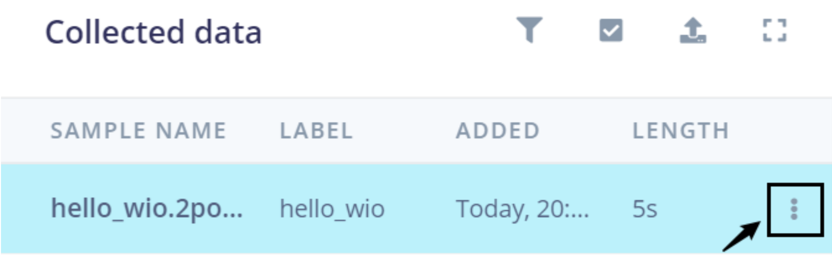


Enter the label, click "Start Sampling", and start saying our keyword over and over again (with some pause in between). Because the recording needs to use SPI Flash which will operate erasing, the time it takes usually longer than we set.

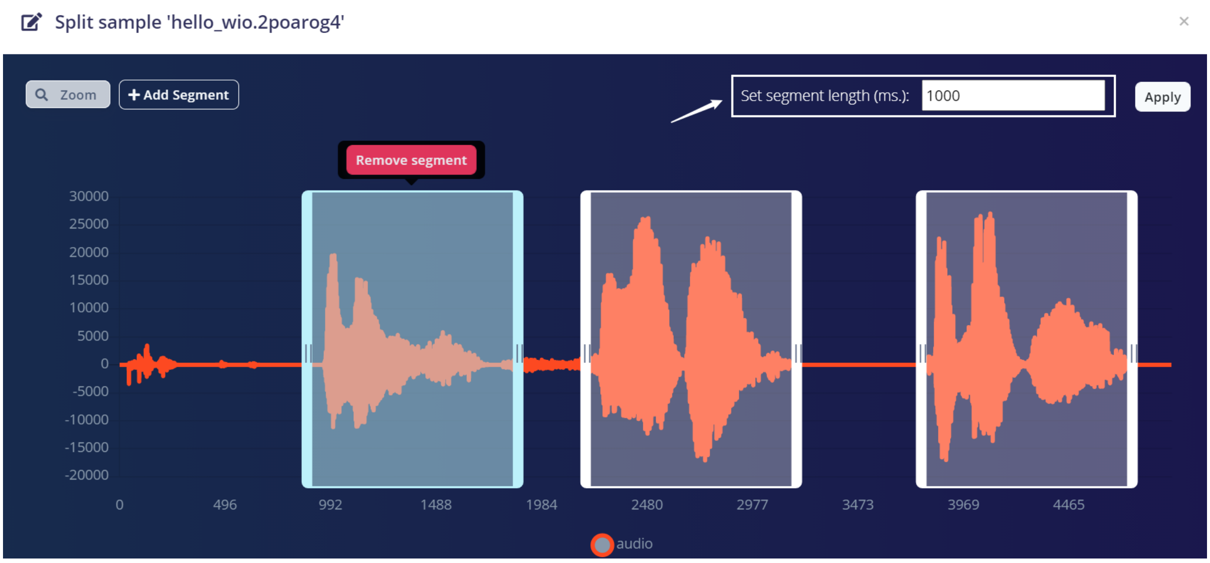
Afterwards, we have a file like this, clearly showing our keywords, separated by some noise. So we can see that I have three. Hello Wio.

This data is not suitable for Machine Learning yet though. We will need to cut out the parts where we say our keyword. This is important because we only want the actual keyword to be labeled as such, and not accidentally label noise, or incomplete sentences.

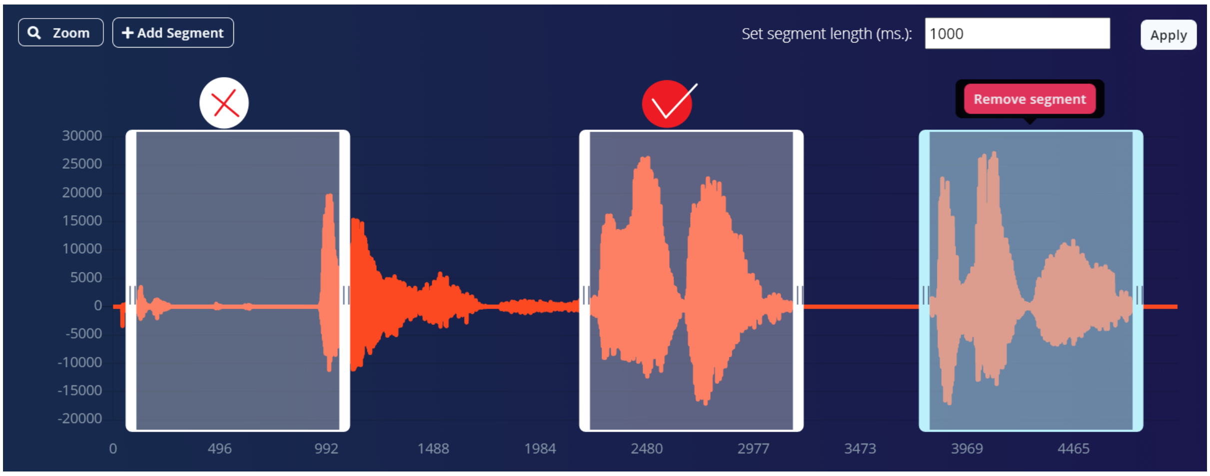
Tap the little three dots here. ⁝ and select the “Split sample”.



we want to look at a certain window length a second here.



That is what we are going to look at. And we need to make sure that the actual word is in there, not noise.



If it has a window like that, there's actually only noise in there, the model gets confused. which is very bad for the accuracy of the model.

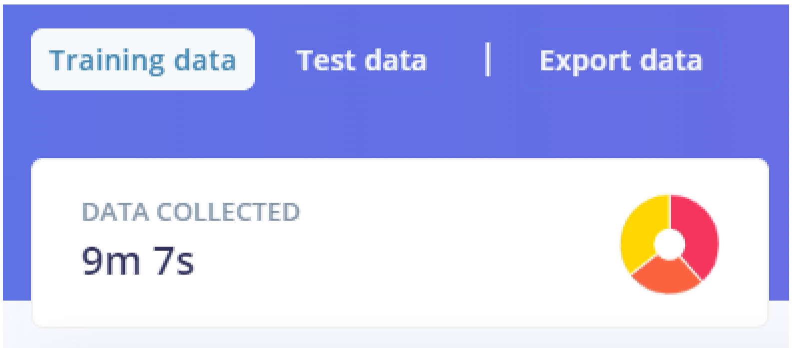
In addition, we can either collect this ourselves or make our life a bit easier by

using a dataset that we get online.

If we search online, we can find some data specially made for keyword spotting. And edge impulse also provides such a dataset.This is a prebuilt dataset for a keyword spotting system based on a subset of data in the Google Speech Commands Dataset, with added noise from the Microsoft Scalable Noisy Speech Dataset. It contains 25 minutes of data per class, split up in 1 second windows, sampled at 16,000Hz.

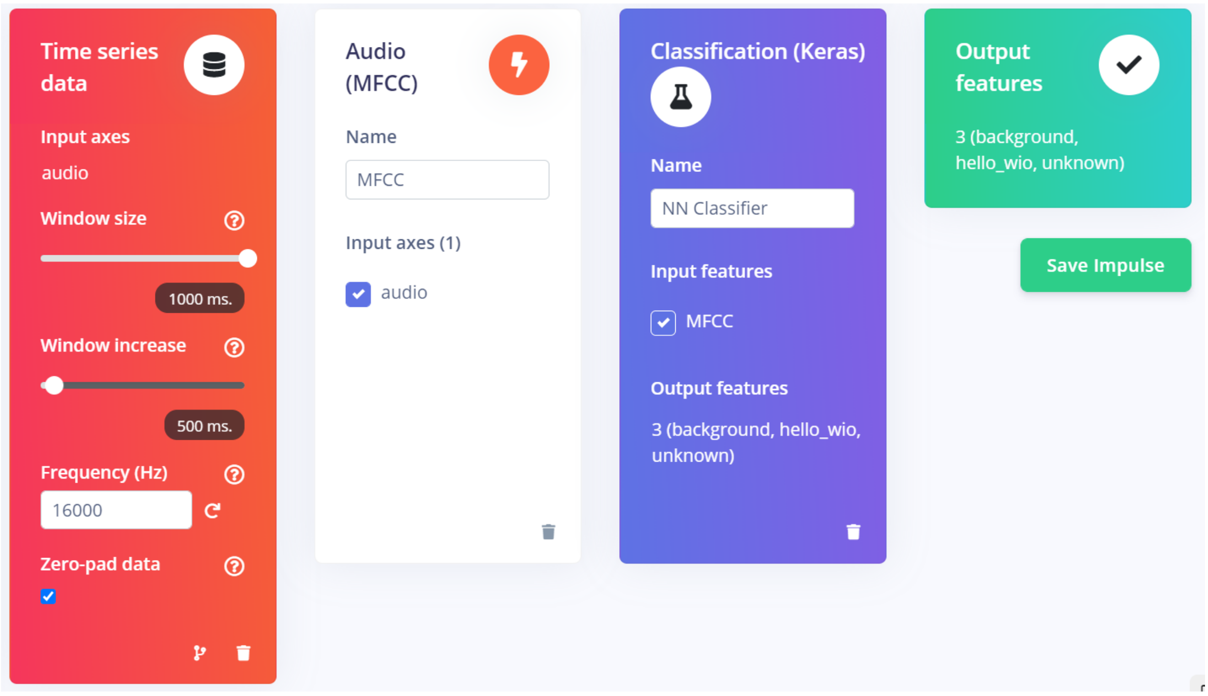
Make sure to capture wide variations of the keyword: leverage our family and our colleagues to help us collect the data, make sure we cover high and low pitches and slow and fast speakers.

Make sure we have a well-balanced dataset.



Impulse Design

With the training set in place, we can design an impulse.

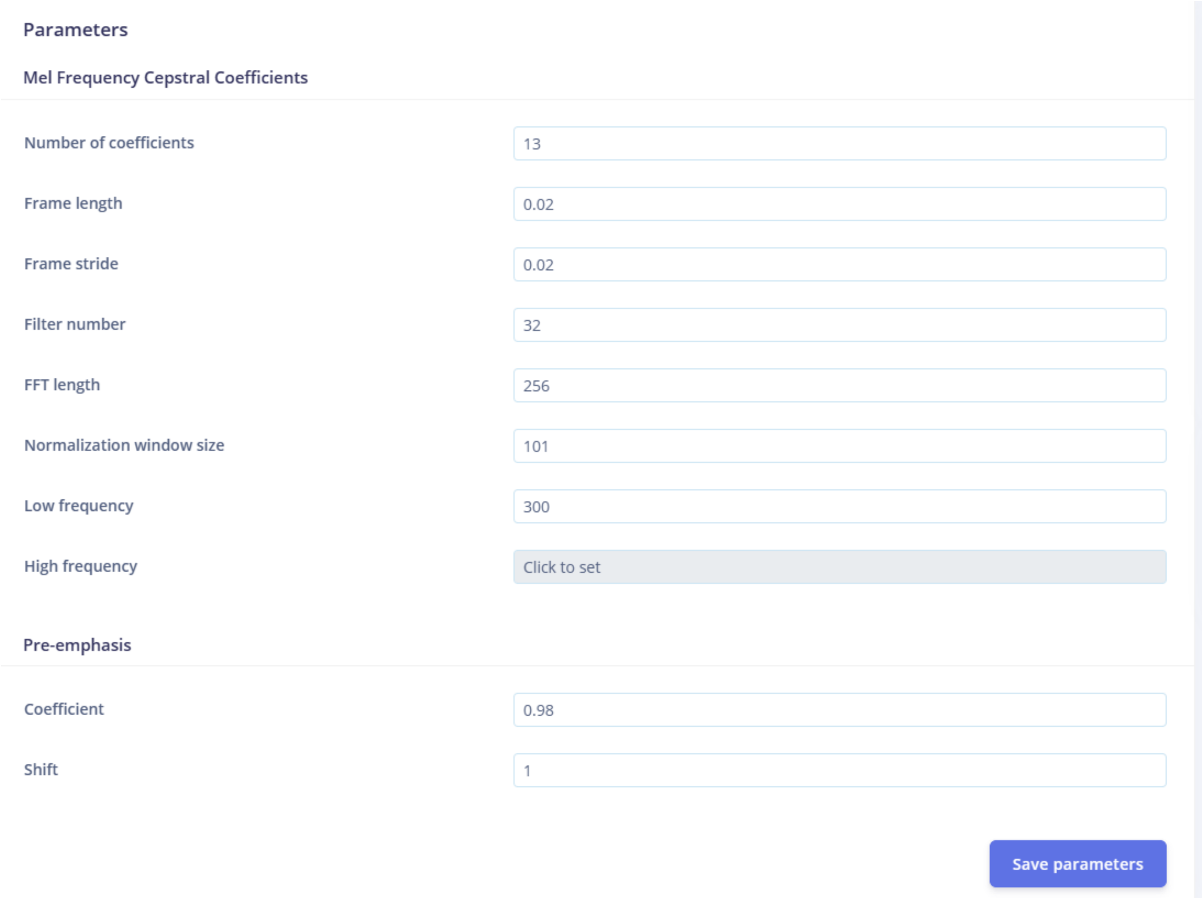


And the pipeline consists of the default settings for time series, data window are correct–1000ms. And the window increase here is not going to be used, because all of our data is already a second long.

Then we add a preprocessing block and we use signal processing to clean up the data before feeding it to the neural network. We have lots of processing blocks for a wide variety of typical senses. We want one specifically for audio. We have the normal spectrogram which is really great for non-voice audio. And then we have an MFE block as well which you can also use for non-voice audio. And here we are dealing with the human voice. So, we'll use the "MFCC" signal processing block.

Feature Extraction--MFCC

MFCC stands for Mel Frequency Cepstral Coefficients. This sounds scary, but it's basically just a way of turning raw audio—which contains a large amount of redundant information—into a simplified form. the "MFCC" block is great for dealing with human speech.



So with all these default parameters set by Edge Impulse for such a project, we won’t change them this time, let's go to generate features.

This is the feature explorer.

So what we see here is all the data in my data sets are shown in three dimensions after the feature extraction step.

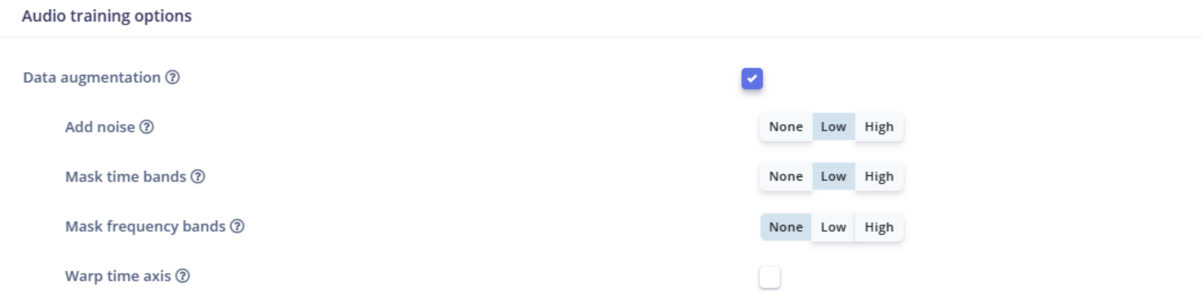
What I am interested in is whether my “Hello Wio”, and my unknown words are nicely separated.

And we see that as a nice separation between the orange clusters, which all contain “Hello Wio” samples and the green cluster contains “unknown” words.

This is a great way to check whether our dataset contains wrong items and to validate whether our dataset is suitable for ML (it should separate nicely).

Model Training--Network:CNN

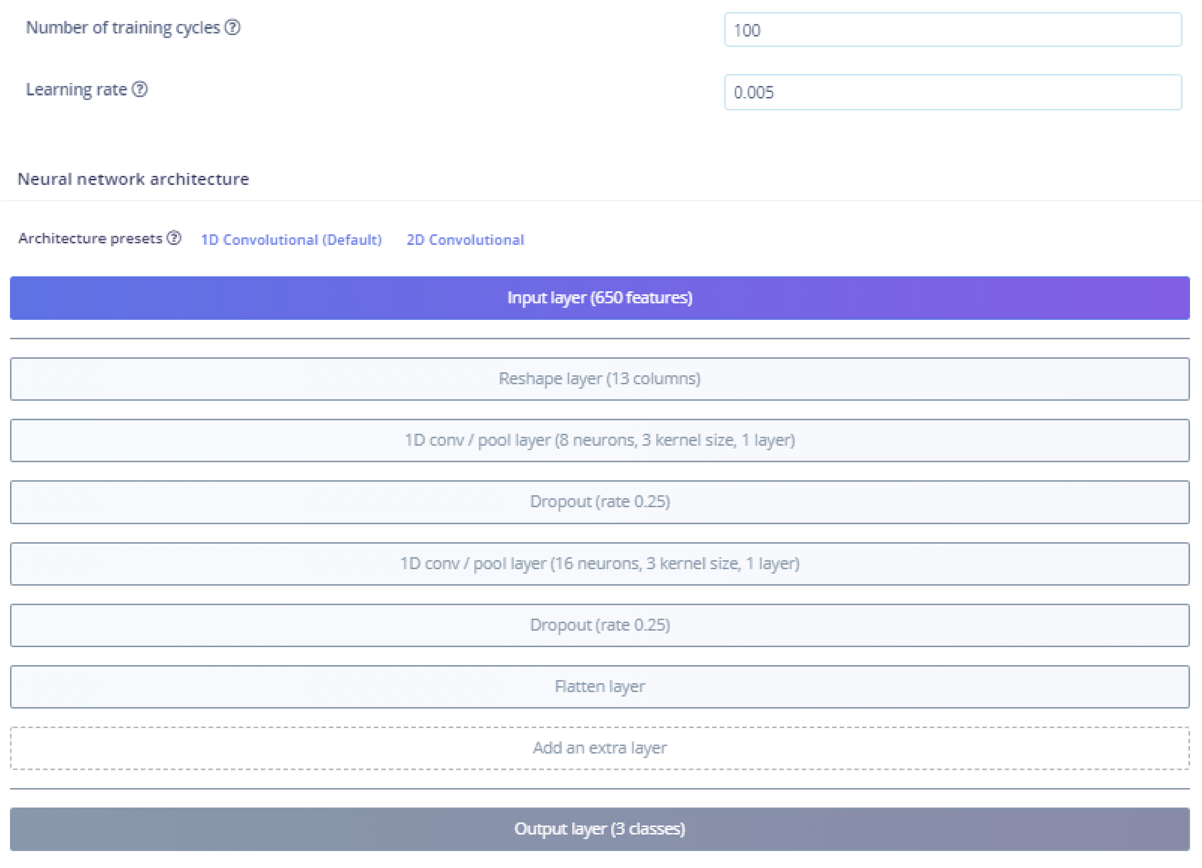
With all data processed it's time to start training a neural network



enable 'Data augmentation', a super-powerful feature where during training we randomly manipulate data in our training data set. So we can add artificial noise to make it more resilient to noisy environments

This is a very quick way to make our dataset work better in real life (with unpredictable sounds coming in) and prevents our neural network from overfitting (as the data samples are changed every training cycle).

We use the default hyperparemeters and neural network provided by Edge Impulse.



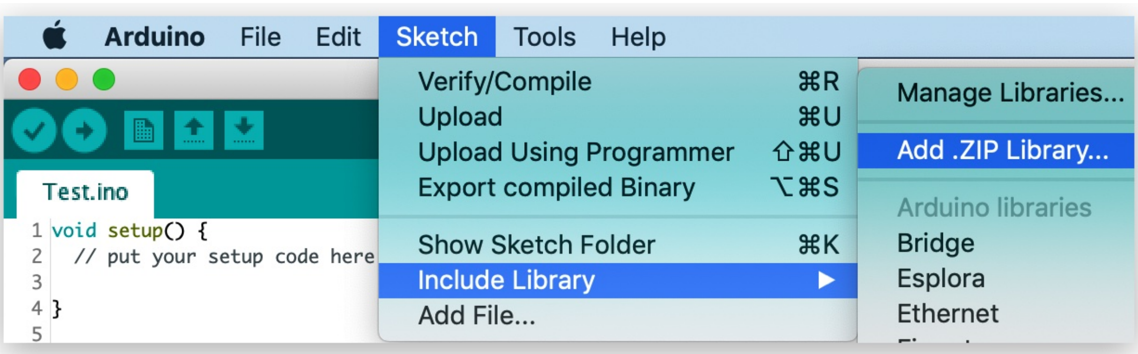
Model Optimization

Model Deployment

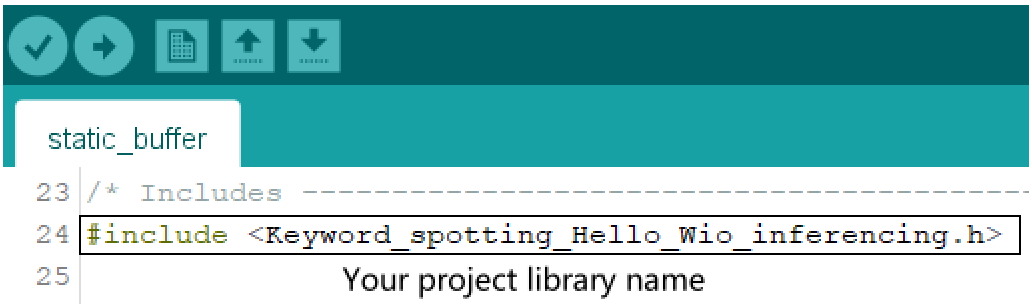
The next step is deployment on the device.

Step 1: After clicking on the Deployment tab, choose Arduino library and download it.

Step 2: Now, the library can be installed to the Arduino IDE. Open the Arduino IDE, click sketch -> Include Library -> Add .ZIP Library, and choose the file that you have just downloaded.



Step 3: Open Examples -> name of your project -> static buffer.



Step 4: Copy the following **Example Code** to replace the original example code:

For Wio Terminal we will rely on DMA or Direct Memory Access controller to obtain samples from ADC (Analog to Digital Converter) and save them to the inference buffer without the involvement of MCU.

That will allow us to collect the sound samples and perform inference at the same time.

```
#include"TFT_eSPI.h"
#include <project_67469_inferencing.h>
enum {ADC_BUF_LEN = 1600};
typedef struct {
  uint16_t btctrl;
  uint16_t btcnt;
  uint32_t srcaddr;
  uint32_t dstaddr;
  uint32_t descaddr;
}dmacdescriptor;
typedef struct {
  signed short *buffers[2];
  unsigned char buf_select;
  unsigned char buf_ready;
  unsigned int buf_count;
  unsigned int n_samples;
}inference_t;
volatile uint8_t recording = 0;
uint16_t adc_buf_0[ADC_BUF_LEN];
uint16_t adc_buf_1[ADC_BUF_LEN];
volatile dmacdescriptor wrb[DMAC_CH_NUM] __attribute__ ((aligned (16)));
dmacdescriptor descriptor_section[DMAC_CH_NUM] __attribute__ ((aligned (16)));
dmacdescriptor descriptor __attribute__ ((aligned (16)));
static inference_t inference;

class FilterBuHp1{
  public:
  FilterBuHp1(){
    v[0] = 0.0;
  }
  private:
  float v[2];
  public:
  float step(float x)
  {
    v[0] = v[1];
    v[1] = (9.621952458291035404e-1f * x) + (0.92439049165820696974f * v[0]);
    return (v[1] - v[0]);
    }
};
FilterBuHp1 filter;

static void audio_rec_callback(uint16_t *buf, uint32_t buf_len) {
  if (recording) {
    for (uint32_t i = 0; i < buf_len; i++) {
      inference.buffers[inference.buf_select][inference.buf_count++] =
filter.step(((int16_t)buf[i] - 1024) * 16);
      if (inference.buf_count >= inference.n_samples) {
        inference.buf_select ^= 1;
        inference.buf_count = 0;
        inference.buf_ready = 1;
```

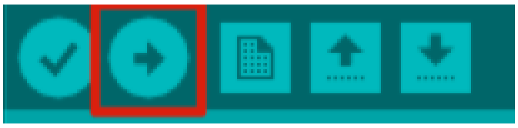
```
      }
    }
  }
}

void DMAC_1_Handler() {
  static uint8_t count = 0;
  if (DMAC->Channel[1].CHINTFLAG.bit.SUSP) {
    DMAC->Channel[1].CHCTRLB.reg = DMAC_CHCTRLB_CMD_RESUME;
    DMAC->Channel[1].CHINTFLAG.bit.SUSP = 1;
    if (count) {
      audio_rec_callback(adc_buf_0, ADC_BUF_LEN);
    }else {
      audio_rec_callback(adc_buf_1, ADC_BUF_LEN);
    }
    count = (count + 1) % 2;
  }
}

void config_dma_adc() {
  DMAC->BASEADDR.reg = (uint32_t)descriptor_section;
  DMAC->WRBADDR.reg = (uint32_t)wrb;
  DMAC->CTRL.reg = DMAC_CTRL_DMAENABLE | DMAC_CTRL_LVLEN(0xf);
  DMAC->Channel[1].CHCTRLA.reg = DMAC_CHCTRLA_TRIGSRC(TC5_DMAC_ID_OVF) |
                                 DMAC_CHCTRLA_TRIGACT_BURST;

  descriptor.descaddr = (uint32_t)&descriptor_section[1];
  descriptor.srcaddr = (uint32_t)&ADC1->RESULT.reg;
  descriptor.dstaddr = (uint32_t)adc_buf_0 + sizeof(uint16_t) * ADC_BUF_LEN;
  descriptor.btcnt = ADC_BUF_LEN;
  descriptor.btctrl = DMAC_BTCTRL_BEATSIZE_HWORD |
                      DMAC_BTCTRL_DSTINC |
                      DMAC_BTCTRL_VALID |
                      DMAC_BTCTRL_BLOCKACT_SUSPEND;
  memcpy(&descriptor_section[0], &descriptor, sizeof(descriptor));

  descriptor.descaddr = (uint32_t)&descriptor_section[0];
  descriptor.srcaddr = (uint32_t)&ADC1->RESULT.reg;
  descriptor.dstaddr = (uint32_t)adc_buf_1 + sizeof(uint16_t) * ADC_BUF_LEN;
  descriptor.btcnt = ADC_BUF_LEN;
  descriptor.btctrl = DMAC_BTCTRL_BEATSIZE_HWORD |
                      DMAC_BTCTRL_DSTINC |
                      DMAC_BTCTRL_VALID |
                      DMAC_BTCTRL_BLOCKACT_SUSPEND;
  memcpy(&descriptor_section[1], &descriptor, sizeof(descriptor));

  NVIC_SetPriority(DMAC_1_IRQn, 0);
  NVIC_EnableIRQ(DMAC_1_IRQn);

  DMAC->Channel[1].CHINTENSET.reg = DMAC_CHINTENSET_SUSP;
```

```
  ADC1->INPUTCTRL.bit.MUXPOS = ADC_INPUTCTRL_MUXPOS_AIN12_Val;
  while (ADC1->SYNCBUSY.bit.INPUTCTRL);
  ADC1->SAMPCTRL.bit.SAMPLEN = 0x00;
  while (ADC1->SYNCBUSY.bit.SAMPCTRL);
  ADC1->CTRLA.reg = ADC_CTRLA_PRESCALER_DIV128;
  ADC1->CTRLB.reg = ADC_CTRLB_RESSEL_12BIT |
                    ADC_CTRLB_FREERUN;
  while (ADC1->SYNCBUSY.bit.CTRLB);
  ADC1->CTRLA.bit.ENABLE = 1;
  while (ADC1->SYNCBUSY.bit.ENABLE);
  ADC1->SWTRIG.bit.START = 1;
  while (ADC1->SYNCBUSY.bit.SWTRIG);
  DMAC->Channel[1].CHCTRLA.bit.ENABLE = 1;
  GCLK->PCHCTRL[TC5_GCLK_ID].reg = GCLK_PCHCTRL_CHEN |
                                   GCLK_PCHCTRL_GEN_GCLK1;
  TC5->COUNT16.WAVE.reg = TC_WAVE_WAVEGEN_MFRQ;
  TC5->COUNT16.CC[0].reg = 3000 - 1;
  while (TC5->COUNT16.SYNCBUSY.bit.CC0);
  TC5->COUNT16.CTRLA.bit.ENABLE = 1;
  while (TC5->COUNT16.SYNCBUSY.bit.ENABLE);
}

static bool microphone_inference_record(void) {
  bool ret = true;
  while (inference.buf_ready == 0) {
    delay(1);
  }
  inference.buf_ready = 0;
  return ret;
}

static int microphone_audio_signal_get_data(size_t offset,
                                             size_t length,
                                             float *out_ptr) {
  numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1][offset],
out_ptr, length);
  return 0;
}

TFT_eSPI tft;
ei_impulse_result_classification_t
currentClassification[EI_CLASSIFIER_LABEL_COUNT];
const char* maxConfidenceLabel;

void runClassifier()
{
  bool m = microphone_inference_record();
  if (!m) {
    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 };
  EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, false);
  if (r != EI_IMPULSE_OK) {
    return;
  }

  float maxValue = 0;
  for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
    ei_impulse_result_classification_t classification_t =
result.classification[ix];
    ei_printf("    %s: %.5f\n", classification_t.label, classification_t.value);
    float value = classification_t.value;
    if (value > maxValue) {
      maxValue = value;
      maxConfidenceLabel = classification_t.label;
    }
    currentClassification[ix] = classification_t;
  }
}

void setup(){
  tft.begin();
  run_classifier_init();
  inference.buffers[0] = (int16_t *)malloc(EI_CLASSIFIER_SLICE_SIZE *
sizeof(int16_t));
  if (inference.buffers[0] == NULL) {
    return;
  }
  inference.buffers[1] = (int16_t *)malloc(EI_CLASSIFIER_SLICE_SIZE *
sizeof(int16_t));
  if (inference.buffers[1] == NULL) {
    free(inference.buffers[0]);
    return;
  }

  inference.buf_select = 0;
  inference.buf_count = 0;
  inference.n_samples = EI_CLASSIFIER_SLICE_SIZE;
  inference.buf_ready = 0;

  config_dma_adc();

  recording = 1;

  tft.setRotation(3);
  tft.setTextSize(4);

}
```

```
void loop(){
  runClassifier();
  if (maxConfidenceLabel == "hello_wio") {
    tft.drawString((String)"Hello Wio", 50, 110);
    delay(3000);
  } else {
    tft.fillScreen(0x0);
  }
}
```

Step 5: Upload the code.



It takes about 5 mins to upload. If the upload is successful, the message "Done uploading." will appear in the status bar.

Step 6: Say “Hello Wio”to the Wio Terminal to see whether it has been woken up.

Reference

Edge Impulse Public project:

https://studio.edgeimpulse.com/public/77128/latest

Practice 4.People counting using Ultrasonic sensor

Project Overview

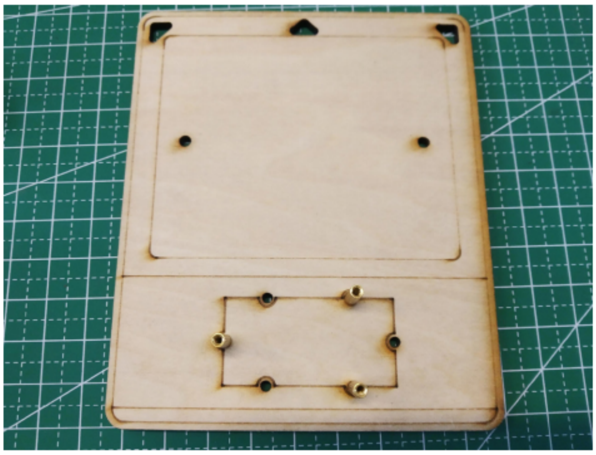
In this project, we will create a people counting system by using Wio Terminal, an ordinary Ultrasonic ranger and a special Deep Learning sauce to top it off and actually make it work.

Material Preparation

Hardware requirements: Wio Terminal

Connection method:

Attach Wio terminal and Ultrasonic sensor with screws to wooden or 3D printed frame, example below:



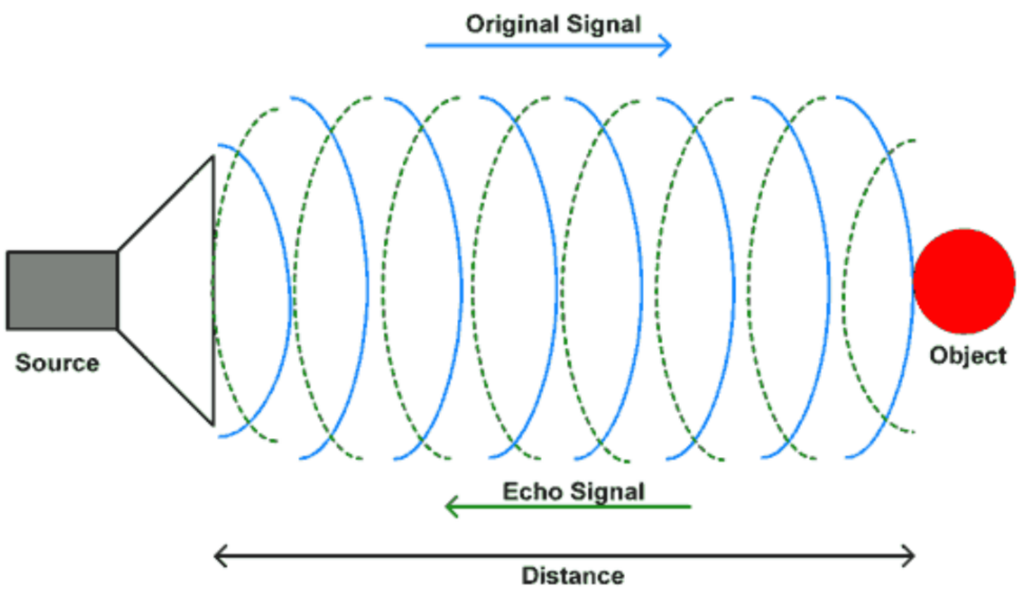
To put the frame on the wall, 3M velcro strips were used.

Additional options include using foam tape, screws or nails.

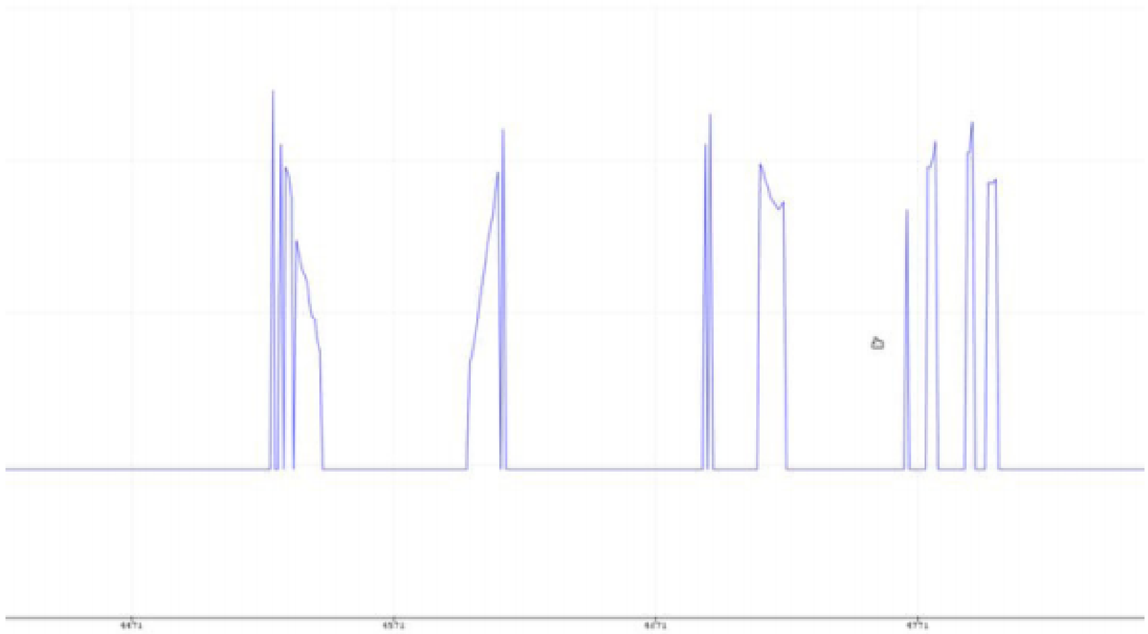
About sensor

First, let's understand the data we can get from the Ultrasonic sensor and how we can utilize it for determining the direction of objects.

This Grove - Ultrasonic ranger is a non-contact distance measurement module that works at 40KHz. When we provide a pulse trigger signal with more than 10uS through the signal pin, the Grove_Ultrasonic_Ranger will issue 8 cycles of 40kHz cycle level and detect the echo. The pulse width of the echo signal is proportional to the measured distance. Here is the formula: Distance =echo signal high time * Sound speed (340M/S)/2.



Now, use this Grove - Ultrasonic ranger. We can immediately see that for walking in, we get relatively high values(corresponding to distance from the object) first, which then decrease. And for walking out, we get completely opposite signal.



Theoretically, we could write an algorithm ourselves by hand, that can determine the direction. Unfortunately, real-life situations are complicated – we have people, that walk fast(shorter curve length) and slow (longer curve length), we have thinner people and people who are... not so thin and so on. So our hand-written algorithm needs to take all of these into account, which will inevitably make it complicated and convoluted. We have a task involving inference signal processing and lots of noisy data with significant variations… And the solution is — Deep Learning.

Warning

Do not hot-plug Grove-Ultrasonic-Ranger, otherwise, it will damage the sensor. The the measured area must be no less than 0.5 square meters and smooth.

Machine Learning Lifecycle

Please Open: https://www.edgeimpulse.com/

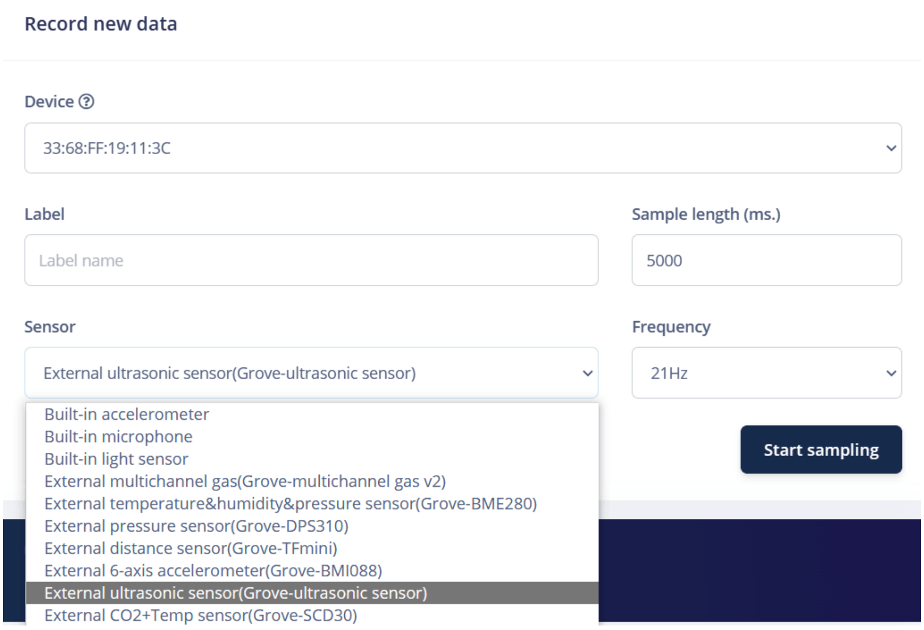
Data Collection

Step1: [Connect Wio Terminal with Edge Impulse]()

Step2: Know what we are going to do

We will train and deploy a simple neural network that can distinguish between people entering or exiting a room using only ultrasonic rangers. So we need to select the sensor we are going to use – Grove-Ultrasonic-Ranger, then know what kind of data we are going to sample --people in and people out.

Select the sensor we are going to use -- Grove-Ultrasonic-Ranger.



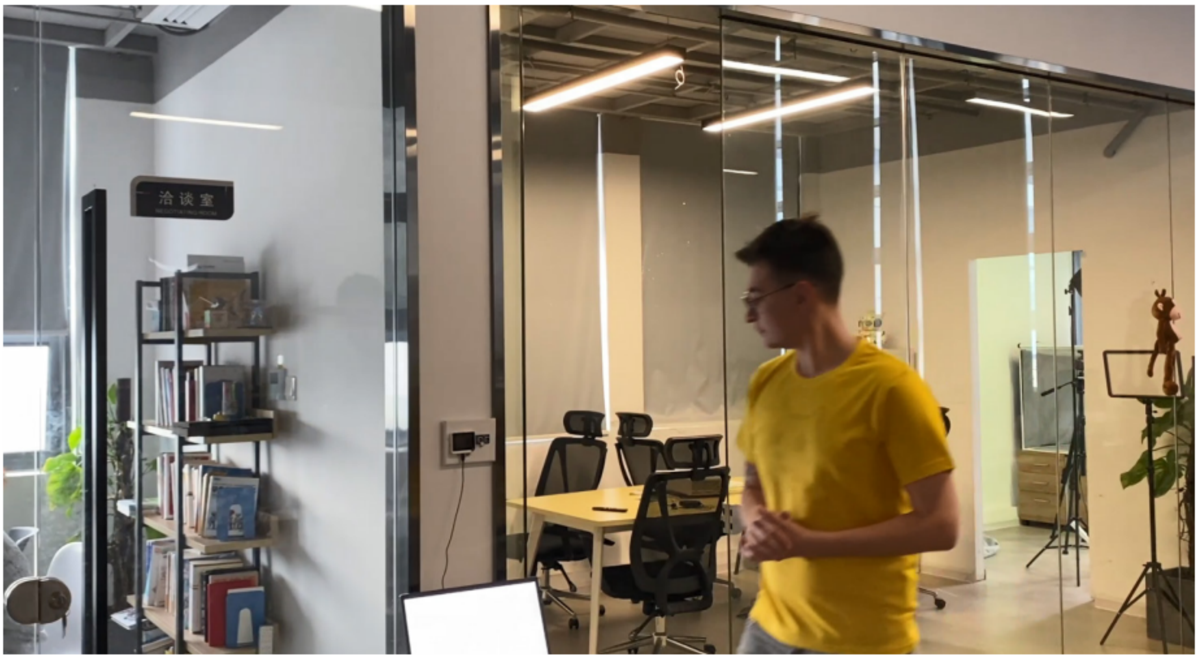
This indicates that we want to record data for 5 seconds (Sample length 5000ms), use a built-in microphone and frequency 21Hz.

Know the data we are going to sample -- people in and people out

Walking in

Walking out

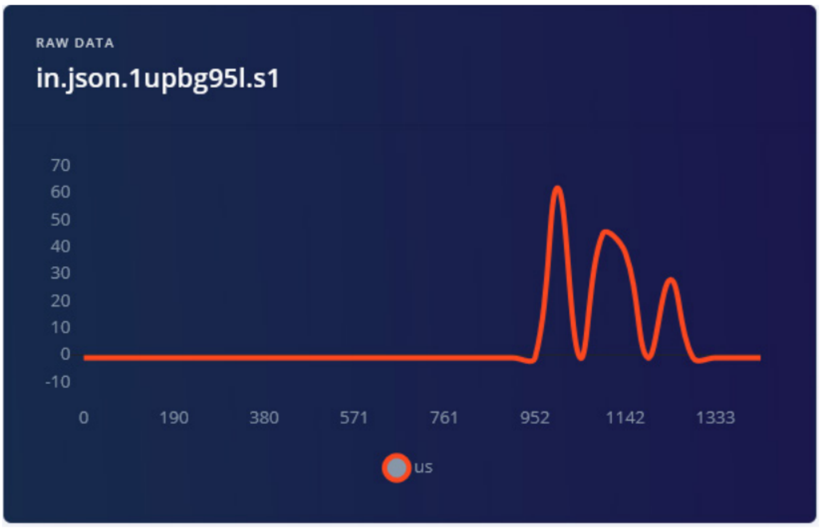
None(walking near the device, not getting closer or further away from it)



Step3: Sample

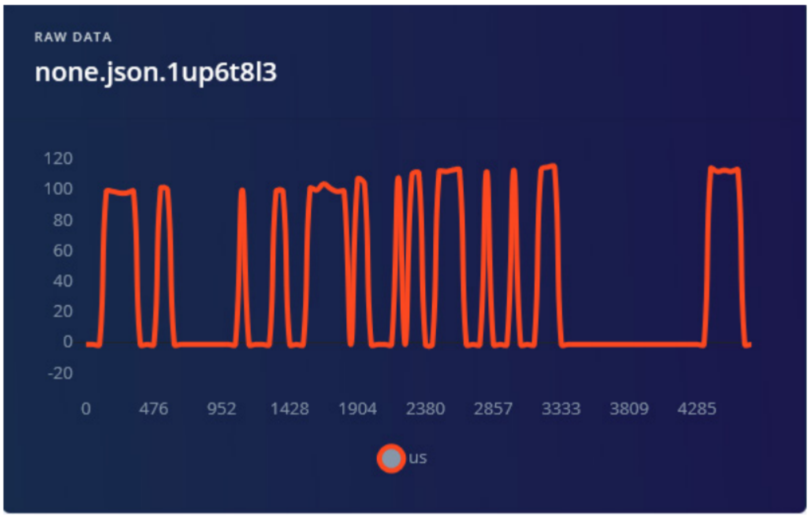
Enter the label, click "Start Sampling",For this lesson, we recorded 1 minute 30 seconds of data for every class, each time recording 5000 ms samples and then cropping them to get 1500 ms samples – remember that variety is very important in the dataset, so make sure you have samples where you (or other people) walk fast, slow, run, etc.

Walking in



Walking out

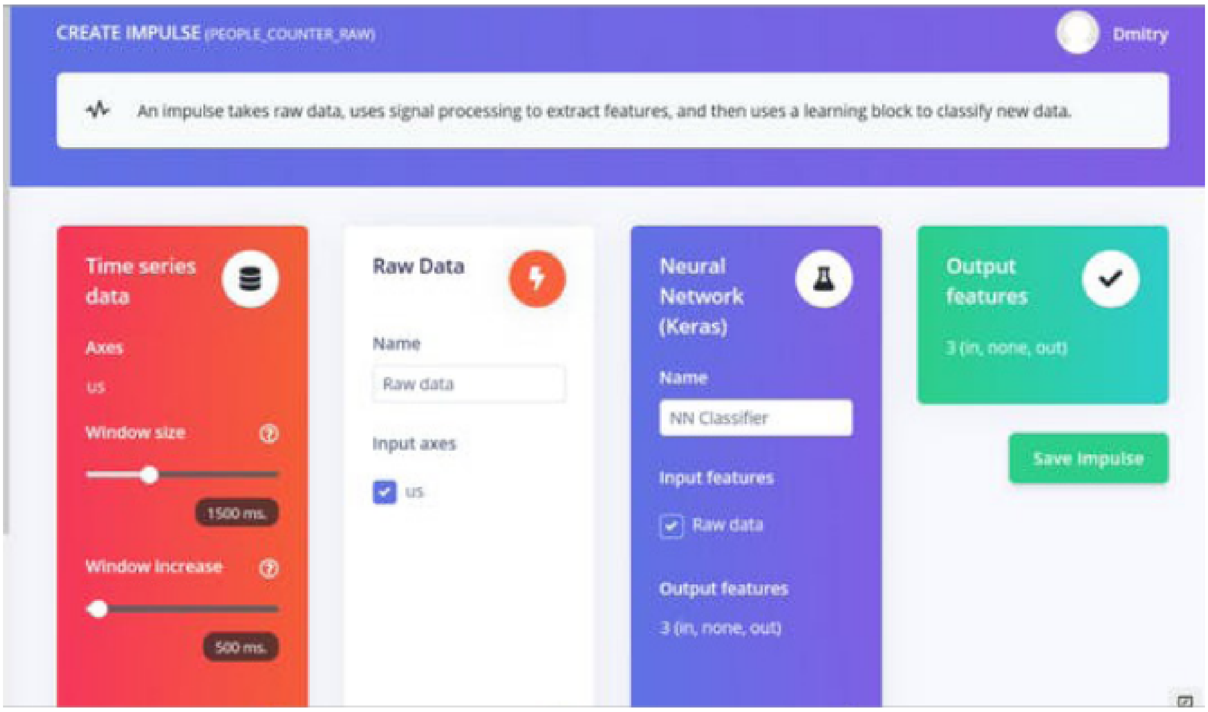
None



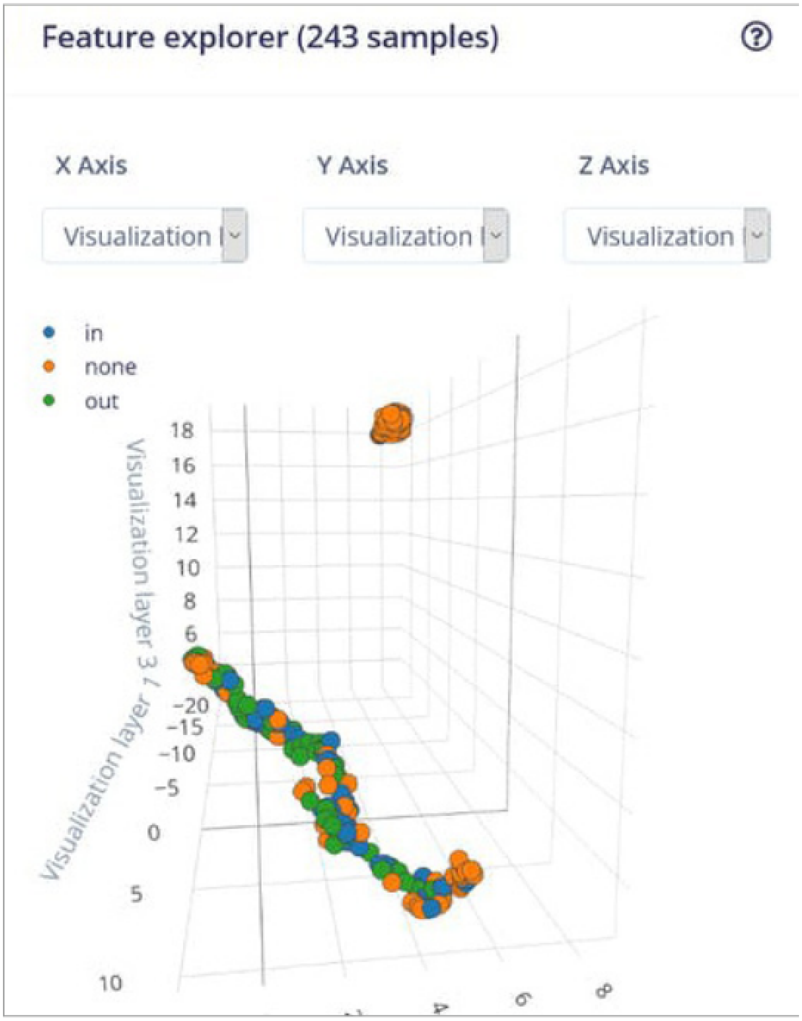
For none category apart from samples that have nobody in front of the device, it is a good idea to include samples that have a person just standing close to the device and walking beside it, to avoid any movement being falsely classified as in or out.

Impulse Design

When we are done with data collection, create your impulse – set window length to 1500 ms and windows size increase to 500 ms.



Feature Extraction--Raw Data

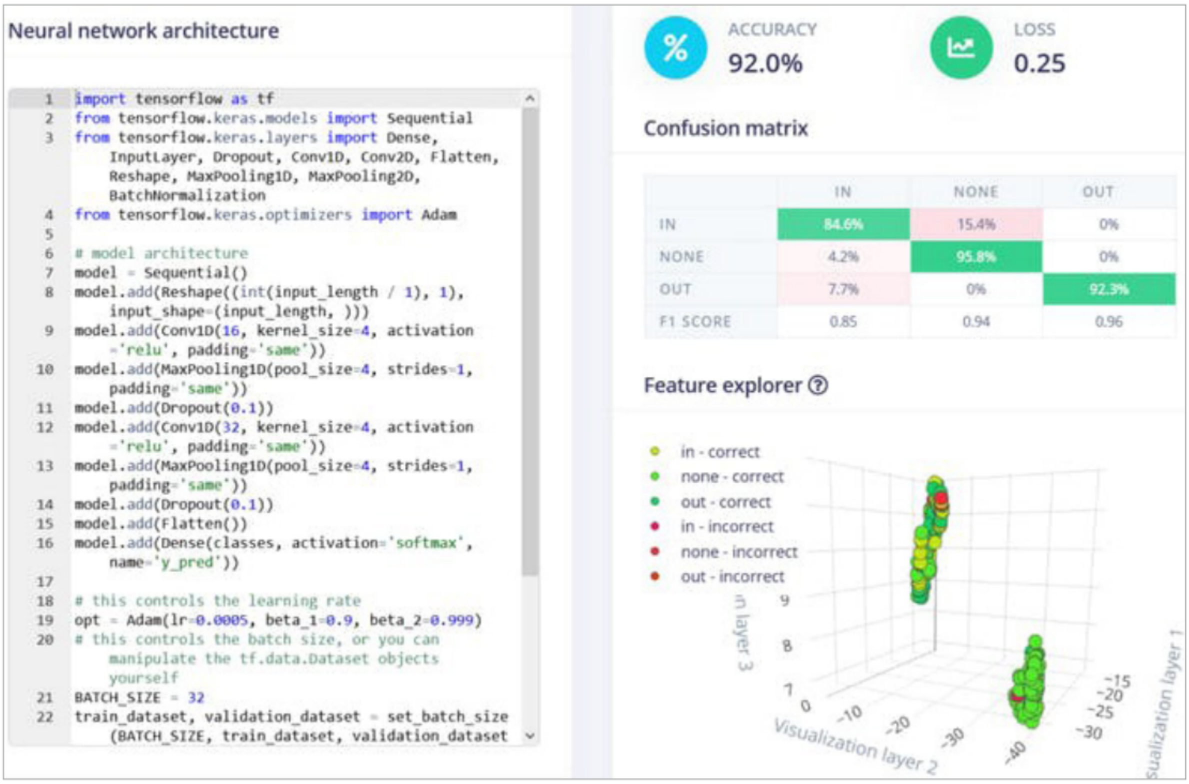


Model Training--Network: CNN

The best results were achieved by tweaking network architecture a bit to

obtain 92% accuracy, for that, you will need to switch to “expert” mode and change

MaxPool1D strides to 1 and pool size to 4.



How good is 92% accuracy and what can be done to improve it?

92% is fairly good as proof of concept or prototype, but horrible as a production model. For production, the mileage may vary – if your application is critical and somehow used in automated control and decision making, you don't really want to have anything below 98 – 99 per cent and even that might be low, think about something like a face recognition system for payment or authentication. Are there ways to improve the accuracy of this system?

The ultrasonic sensor is a cheap and ubiquitous sensor, but it is relatively slow and not very precise. We can get better data by using Grove TF Mini LiDAR Module.

• Get more data and possibly place the sensor lower, at normal human waist level to make sure it can detect shorter than normal height people and children.

• Two are better than one – having two sensors taking measurements at slightly different places will not add too much data (we only have 31 data points in each sample) but might increase the accuracy. To explore more interesting ideas, a built-in light sensor can be used if Wio Terminal is appropriately located.

Once the model is trained we can perform live classification with data from the device – here we found that a window size increase of 500 ms actually doesn't work that well and produces more false positives, so at the next step when deploying to the device, it is better to increase the value to 750 ms.

Model Optimization

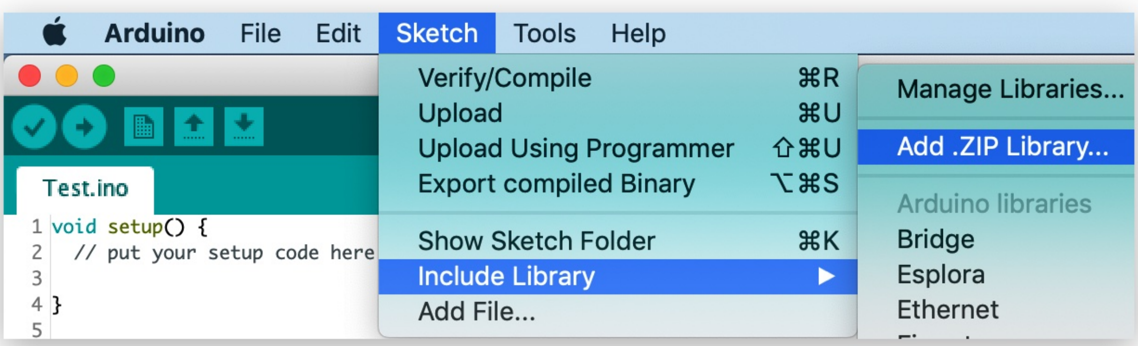
Model Deployment

The next step is deployment on the device.

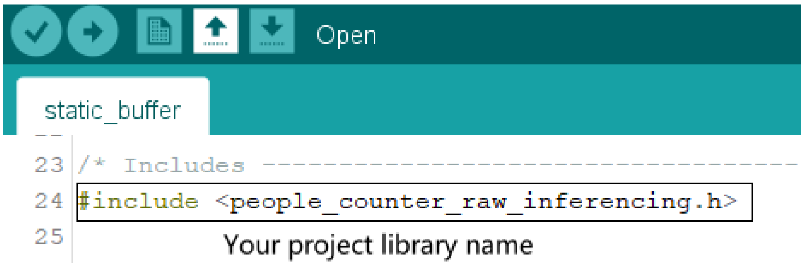
Step 1: After clicking on the Deployment tab, choose Arduino library and download it.

Step 2: Now, the library can be installed to the Arduino IDE. Open the Arduino IDE, click sketch -> Include Library -> Add .ZIP Library, and choose the file that you have just

downloaded.



Step 3: Open Examples -> name of your project -> static buffer.



Step 4: Copy the following **Example Code** to replace the original example code:

This time we will be using continuous inference examples to make sure we are not missing any important data.

```
#include <people_counter_inferencing.h>
#include <Seeed_Arduino_FreeRTOS.h>
#include "Ultrasonic.h"
#include "TFT_eSPI.h"
#include <lvgl.h>

#define ERROR_LED_LIGHTUP_STATE HIGH
#define LVGL_TICK_PERIOD 10

/* 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 = 500;
static float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = {0};
static float inference_buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE];
float distance;

uint8_t axis_num = 1;
int16_t peopleCount = 0;
uint16_t peopleIn = 0;
uint16_t peopleOut = 0;
```

```
lv_obj_t *LogOutput;
lv_obj_t *peopleInLabel;
lv_obj_t *peopleOutLabel;
lv_obj_t *peopleNumLabel;

const char *prev_prediction = "none";

TaskHandle_t Handle_aTask;
TaskHandle_t Handle_bTask;
TaskHandle_t Handle_cTask;

Ultrasonic ultrasonic(0);
TFT_eSPI tft;
static lv_disp_buf_t disp_buf;
static lv_color_t buf[LV_HOR_RES_MAX * 10];

/**
* @brief      Arduino setup function
*/

void setup()
{
    pinMode(WIO_KEY_A, INPUT_PULLUP);
    pinMode(WIO_KEY_B, INPUT_PULLUP);
    pinMode(WIO_KEY_C, INPUT_PULLUP);

    lv_init();

    tft.begin();
    tft.setRotation(3);
    // put your setup code here, to run once:
    Serial.begin(115200);

    lv_disp_buf_init(&disp_buf, buf, NULL, LV_HOR_RES_MAX * 10);
    lv_disp_drv_t disp_drv;
    lv_disp_drv_init(&disp_drv);
    disp_drv.hor_res = 320;
    disp_drv.ver_res = 240;
    disp_drv.flush_cb = my_disp_flush;
    disp_drv.buffer = &disp_buf;
    lv_disp_drv_register(&disp_drv);

    lv_buttons();
    ////////////////////
    // Enter configuration mode

    if (EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME != axis_num) {
        ei_printf("ERR: EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME should be equal to
(%d) (the (%d) sensor axes)\n", axis_num, axis_num);
        return;
    }
```

```
    vSetErrorLed(LED_BUILTIN, ERROR_LED_LIGHTUP_STATE);

    // Create the threads that will be managed by the rtos
    // Sets the stack size and priority of each task
    // Also initializes a handler pointer to each task, which are important to
communicate with and retrieve info from tasks

    xTaskCreate(lv_tick_task, "LVGL Tick", 128, NULL, tskIDLE_PRIORITY + 1,
&Handle_aTask);
    xTaskCreate(run_inference_background,"Inference", 512, NULL,
tskIDLE_PRIORITY + 1, &Handle_bTask);
    xTaskCreate(read_data, "Data collection", 256, NULL, tskIDLE_PRIORITY + 2,
&Handle_cTask);

    // Start the RTOS, this function will never return and will schedule the
tasks.
    vTaskStartScheduler();
}

/**
* @brief      Printf function uses vsnprintf and output using Arduino Serial
*
* @param[in]  format     Variable argument list
*/
void update_screen()
{
  peopleCount = peopleIn - peopleOut;
  lv_label_set_text_fmt(peopleInLabel, "%d", peopleIn);
  lv_label_set_text_fmt(peopleOutLabel, "%d", peopleOut);
  lv_label_set_text_fmt(peopleNumLabel, "%d", peopleCount);
  lv_task_handler();
}

static void lv_tick_task(void* pvParameters) {
    while(1){
      lv_tick_inc(LVGL_TICK_PERIOD);
      delay(LVGL_TICK_PERIOD);
    }
}

static void DisplayPrintf(const char* format, ...)
{
    va_list arg;
    va_start(arg, format);
    String str{StringVFormat(format, arg)};
    va_end(arg);

    Log("%s\n", str.c_str());
    lv_label_set_text(LogOutput, str.c_str());
    lv_task_handler();
}
```

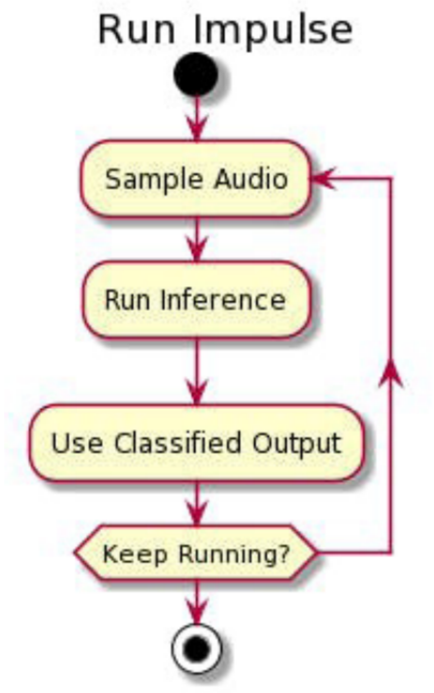
```
void my_disp_flush(lv_disp_drv_t *disp, const lv_area_t *area, lv_color_t
*color_p)
{
  uint16_t c;

  tft.startWrite(); /* Start new TFT transaction */
  tft.setAddrWindow(area->x1, area->y1, (area->x2 - area->x1 + 1), (area->y2 -
area->y1 + 1)); /* set the working window */
  for (int y = area->y1; y <= area->y2; y++) {
    for (int x = area->x1; x <= area->x2; x++) {
      c = color_p->full;
      tft.writeColor(c, 1);
      color_p++;
    }
  }
  tft.endWrite(); /* terminate TFT transaction */
  lv_disp_flush_ready(disp); /* tell lvgl that flushing is done */
}

void lv_buttons(void)
{
    lv_obj_t *peopleInDisplay = lv_btn_create(lv_scr_act(), NULL);     /*Add a
button the current screen*/
    lv_obj_set_pos(peopleInDisplay, 20, 60);                            /*Set
its position*/
    lv_obj_set_size(peopleInDisplay, 120, 50);                          /*Set
its size*/
    peopleInLabel = lv_label_create(peopleInDisplay, NULL);          /*Add a
label to the button*/
    lv_label_set_text(peopleInLabel, "0");                     /*Set the labels
text*/
    lv_obj_t *peopleOutDisplay = lv_btn_create(lv_scr_act(), NULL);     /*Add a
button the current screen*/
    lv_obj_set_pos(peopleOutDisplay, 180, 60);                            /*Set
its position*/
    lv_obj_set_size(peopleOutDisplay, 120, 50);                          /*Set
its size*/
    peopleOutLabel = lv_label_create(peopleOutDisplay, NULL);          /*Add a
label to the button*/
    lv_label_set_text(peopleOutLabel, "0");                     /*Set the labels
text*/
    lv_obj_t *peopleNumDisplay = lv_btn_create(lv_scr_act(), NULL);     /*Add a
button the current screen*/
    lv_obj_set_pos(peopleNumDisplay, 90, 160);                            /*Set
its position*/
    lv_obj_set_size(peopleNumDisplay, 140, 70);                          /*Set
its size*/
    peopleNumLabel = lv_label_create(peopleNumDisplay, NULL);          /*Add a
label to the button*/
    lv_label_set_text(peopleNumLabel, "0");                     /*Set the labels
```

```
text*/
    LogOutput = lv_label_create(lv_scr_act(), NULL);
    lv_label_set_long_mode(LogOutput, LV_LABEL_LONG_BREAK);     /*Break the long
lines*/
    lv_label_set_recolor(LogOutput, true);                      /*Enable
re-coloring by commands in the text*/
    lv_label_set_align(LogOutput, LV_LABEL_ALIGN_LEFT);       /*Center aligned
lines*/
    lv_obj_set_width(LogOutput, 320);
    lv_obj_align(LogOutput, NULL, LV_ALIGN_IN_TOP_LEFT, 20, 10);
}

#define DLM "\r\n"

static String StringVFormat(const char* format, va_list arg)
{
    const int len = vsnprintf(nullptr, 0, format, arg);
    char str[len + 1];
    vsnprintf(str, sizeof(str), format, arg);
    return String{str};
}

static void Abort(const char* format, ...)
{
    va_list arg;
    va_start(arg, format);
    String str{ StringVFormat(format, arg) };
    va_end(arg);
    Serial.printf("ABORT: %s" DLM, str.c_str());
    while (true) {}
}

static void Log(const char* format, ...)
{
    va_list arg;
    va_start(arg, format);
    String str{StringVFormat(format, arg)};
    va_end(arg);
    Serial.print(str);
}

/**
 * @brief      Run inferencing in the background.
 */
static void run_inference_background(void* pvParameters)
{
    // 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, 3 /* no. of readings */, 2 /* min.
readings the same */, 0.6 /* 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) {
            Log("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) {
            Log("ERR: Failed to run classifier (%d)\n", err);
            return;
        }
        // print the predictions
        Log("Predictions ");
        Log("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
            result.timing.dsp, result.timing.classification,
result.timing.anomaly);
        Log(": ");
        // ei_classifier_smooth_update yields the predicted label
        const char *prediction = ei_classifier_smooth_update(&smooth, &result);
        Log("%s ", prediction);
        if (prediction != prev_prediction)
        {
        if (prediction == "out") {peopleOut++; DisplayPrintf("#ff00ff Person
left#");}
        if (prediction == "in") {peopleIn++; DisplayPrintf("#0000ff Person
entered#");}
        prev_prediction = prediction;
        update_screen();
        }
        // print the cumulative results
        Log(" [ ");
        for (size_t ix = 0; ix < smooth.count_size; ix++) {
            Log("%u", smooth.count[ix]);
            if (ix != smooth.count_size + 1) {
                Log(", ");
            }
            else {
              Log(" ");
```

```
            }
        }
        Log("]\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
*/
static void read_data(void* pvParameters)
{
    while (1) {
        // Determine the next tick (and then sleep later)
        uint64_t next_tick = micros() + (EI_CLASSIFIER_INTERVAL_MS * 1000);
        // roll the buffer -axis_num points so we can overwrite the last one
        numpy::roll(buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, -axis_num);
        distance = ultrasonic.MeasureInCentimeters();
        if (distance > 200.0) { distance = -1;}
        buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 1] = distance;
        // 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);
    }
}

void loop()
{
  //nothing, all the work is done in two threads
}
```

If you remember, in the first Practice, for the inference, we would collect all the data points in the sample, perform the inference and then go back to sampling – that means that when feeding the data to the neural network we would pause the data collection and lose some of the data.



That is not optimal and we can use either DMA (Direct Memory Access), threading or multiprocessing to fix this issue.

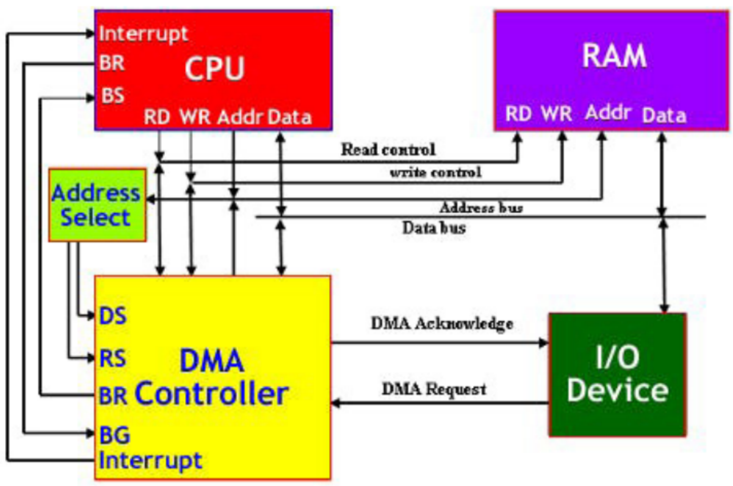
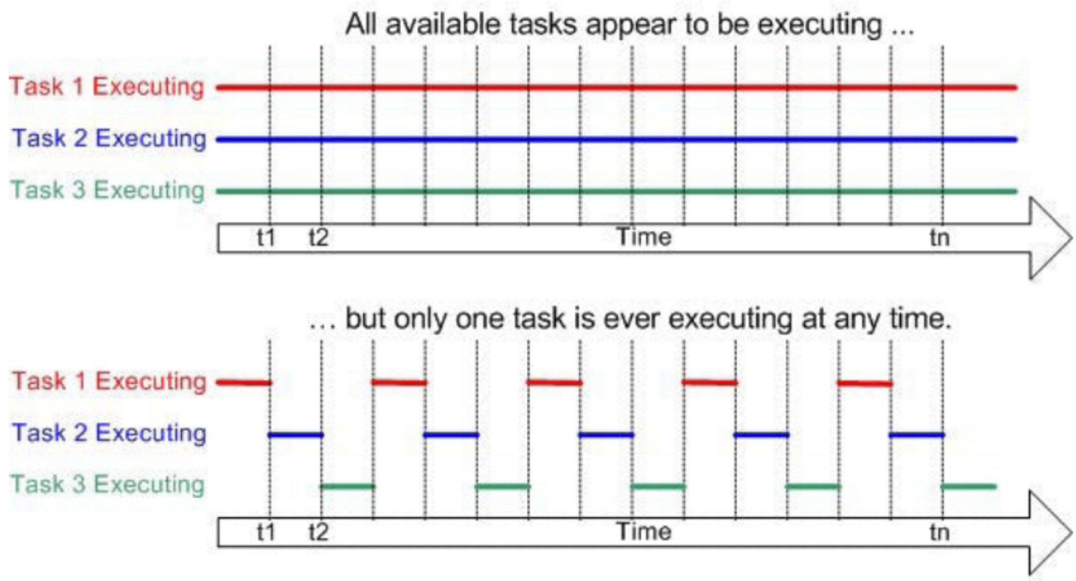


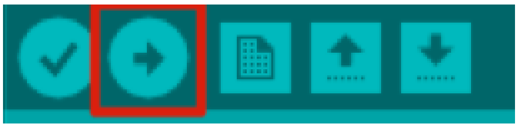
Fig: Showing DMA Mode of Data Transfer

Wio Terminal MCU (Cortex M4F core) only has one core, so multiprocessing is not an option – so in this case, we will use FreeRTOS and threads. What is going to happen is that during the inference process, FreeRTOS will pause inference for a brief moment, collect the data sample and then go back to inference



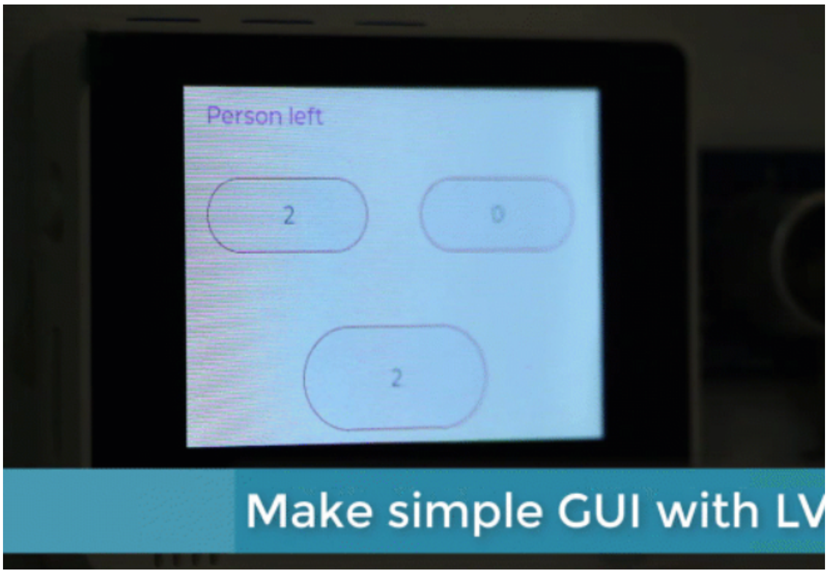
This way the actual inference will take a little longer, but the difference is negligible for this particular use case. We perform inference every 500 ms, so every 500 ms slice of the time window will be performed inference on for 3 times. Then we take the result that has the highest confidence across 3 inferences – for example, we have the highest confidence for "out" label 2 times and for "none" label one time, thus the result should be classified as "out". To simplify the testing we will add the lines that turn on Wio Terminal screen when a person is entering the room and turn it off when a person exits.

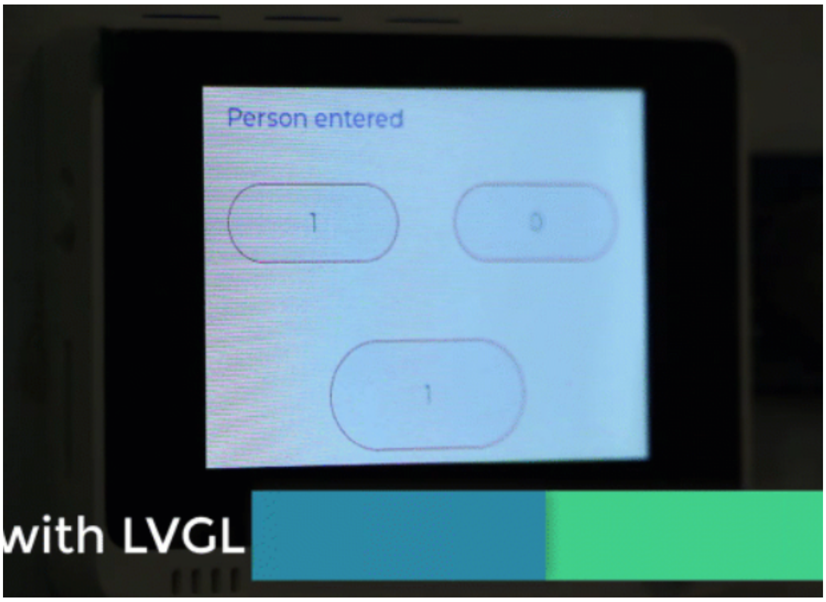
Step 5: Upload the code.



It takes about 5 mins to upload. If the upload is successful, the message "Done uploading." will appear in the status bar.

Step 6:





Reference

Edge Impulse Public project

https://studio.edgeimpulse.com/public/18808/latest

Practice 5. Anomaly detection using Grove BME280

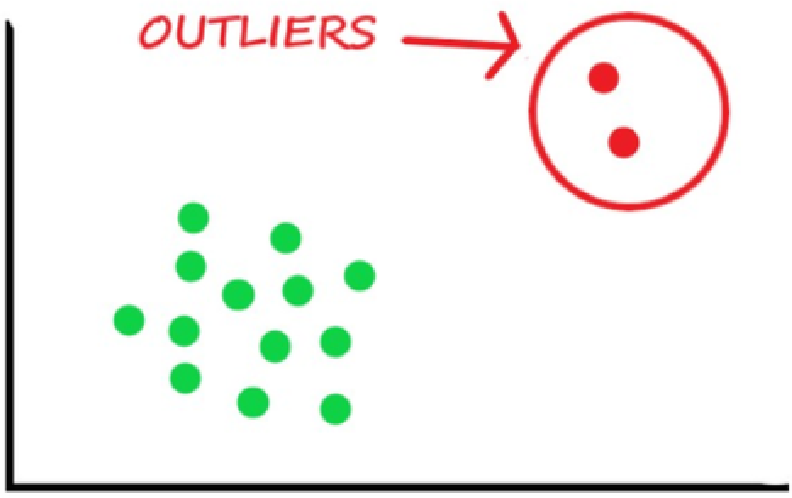
Project Overview

In this project, we will use data from BME280, perform anomaly detection on-device.

Anomalies. Or specifically anomaly detection for predictive maintenance.

Some workshops will have requirements for specific range of temperature, humidity and air pressure because the abnormal environments will have adverse effects on their products. Similarly, greenhouse planting, the breeding hatchery has requirements for these three indices, a good environment helps its planting and hatching.

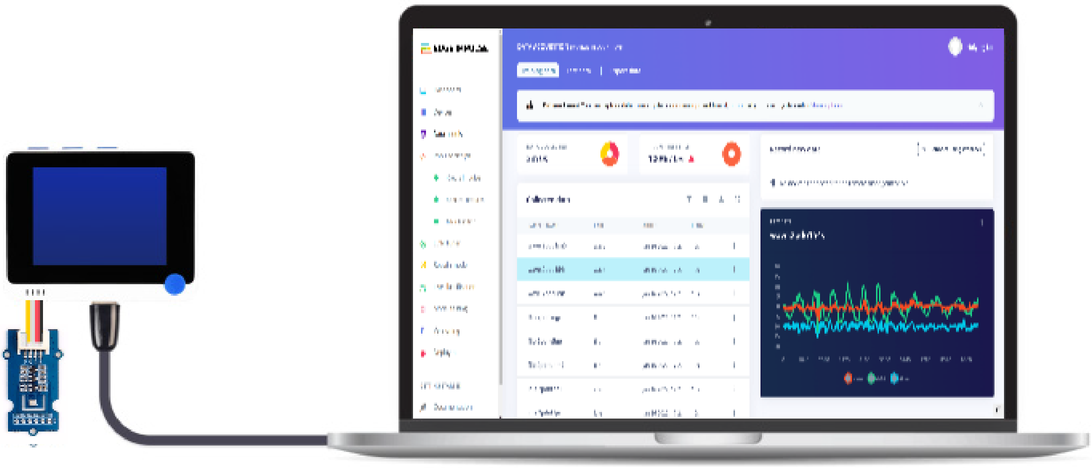
In these situations, we really just want our model to be able to interpret all the data as "normal" and "abnormal". It doesn't matter what are the exact characteristics of "abnormal" – they can be wildly different, the important thing is, if the "abnormal" class is detected, some measures need to be implemented. What I described now is the premise behind using Machine Learning for predictive maintenance. We monitor the state of the device or a place, be it an air conditioner, water pump or other machinery with sensors and based on the profile of known "normal" operation, try to detect when something goes SLIGHTLY wrong before it goes SERIOUSLY wrong.



Material Preparation

Hardware requirements: Wio Terminal

Connection method:



About sensor

Grove BME280 provides a precise measurement of not only barometric pressure and temperature, but also the humidity in the environment. The air pressure can be measured in a range from 300 hPa to 1100hPa with ±1.0 hPa accuracy, while the sensor works perfectly for temperatures between - 40℃ and 85℃ with an accuracy of ±1℃. As for the humidity, you can get a humidity value with an error of less than 3%.

Owing to its high accuracy in measuring the pressure, and the pressure changes with altitude, we can calculate the altitude with ±1 meter accuracy, which makes it a precise altimeter as well.

Machine Learning Lifecycle

Please Open: https://www.edgeimpulse.com/

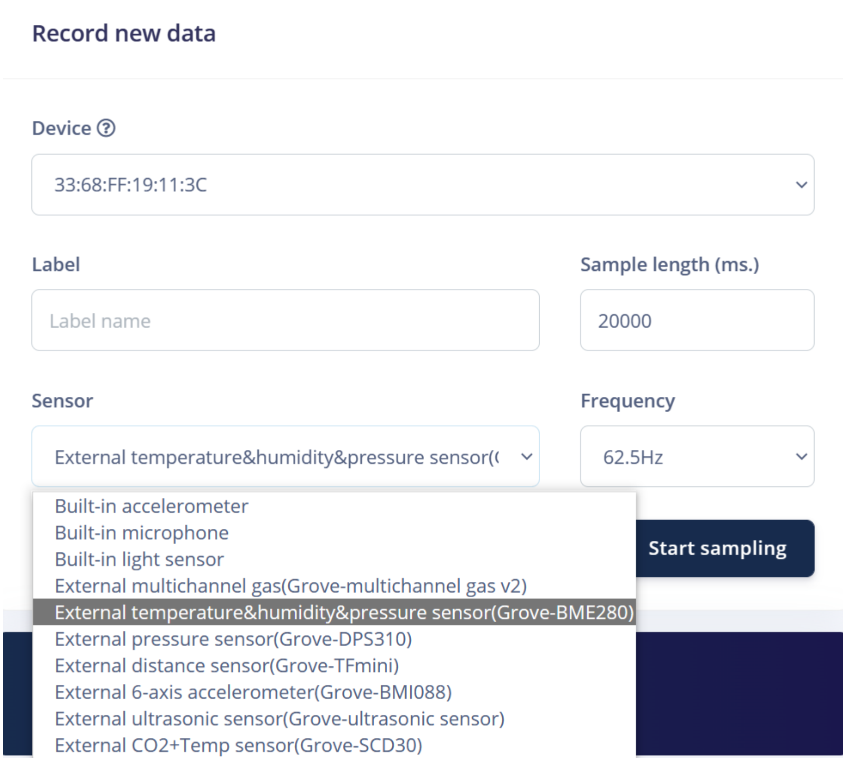
Data Collection

Step1: Connect Wio Terminal with Edge Impulse

Step2: Know what we are going to do

We will train and deploy a simple neural network that is able to interpret all the data as "normal" and "abnormal" using BME280. So we need to select the sensor we are going to use – Grove-BME280, then know what kind of data we are going to sample --data of normal state.

Select the sensor we are going to use -- Grove-BME280.



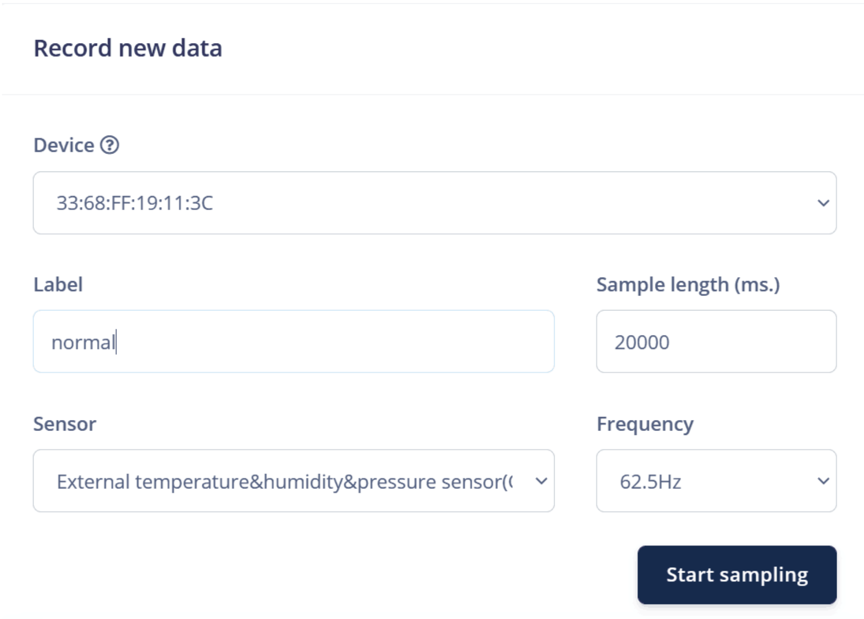
This indicates that we want to record data for 20 seconds (Sample length 20000ms), use Grove-BME280 and frequency 62.5Hz.

Know the data we are going to sample

The workshops have requirements for specific range of temperature, humidity and air pressure because the abnormal environments will have adverse effects on their products.

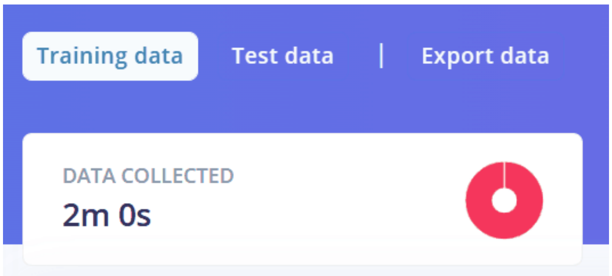
We want to sample data that is in its normal state.

Step3: Sample



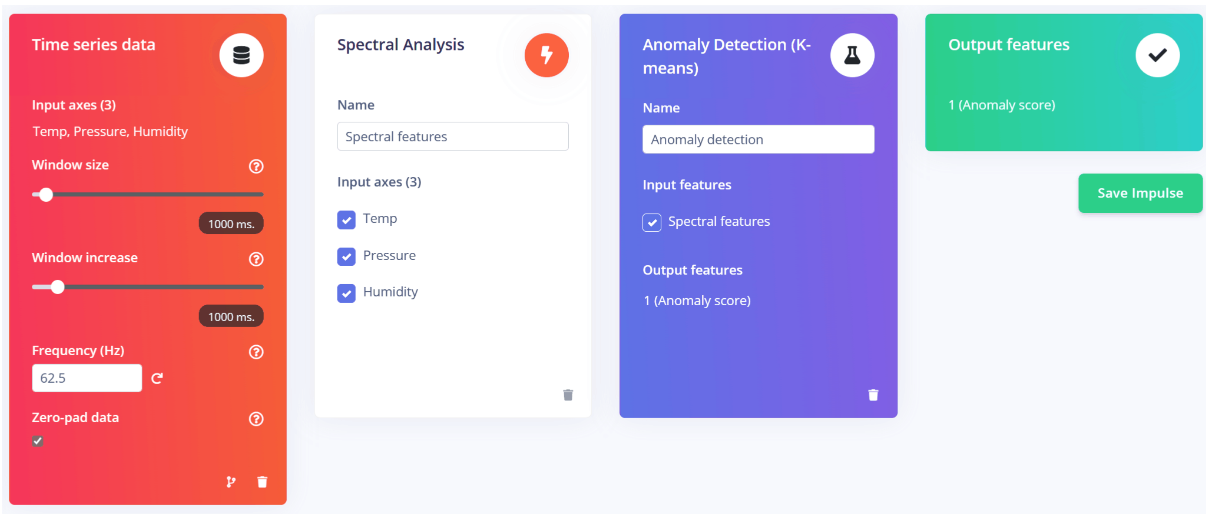
Enter the label, click "Start Sampling".

Now, we have recorded around 2 minutes of data:



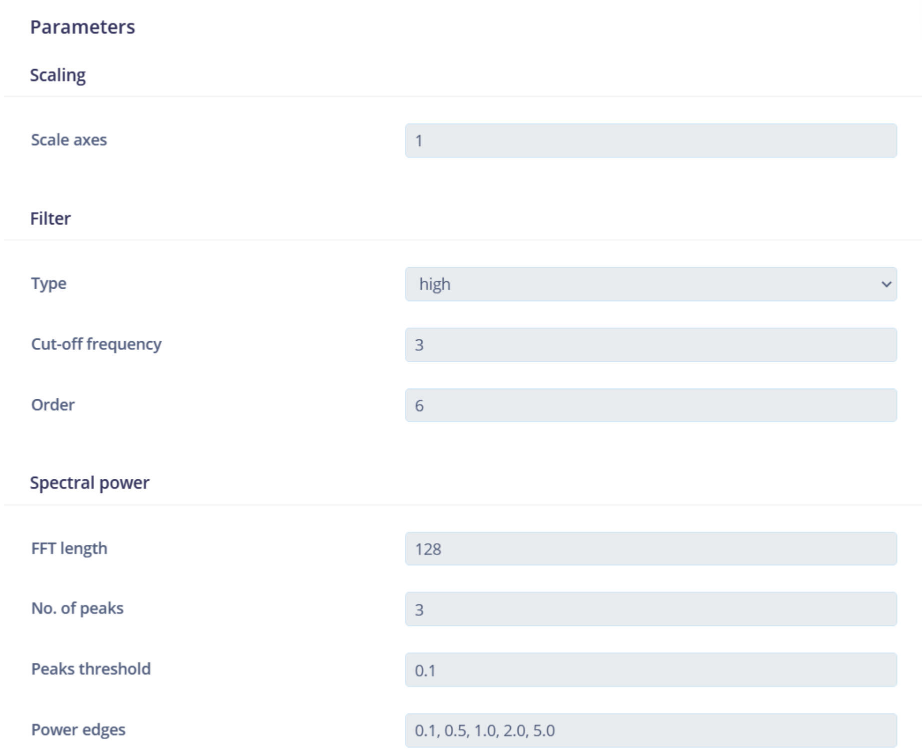
Impulse Design

When we are done with data collection, create our impulse – set window length to 1000 ms and windows size increase to 1000 ms.

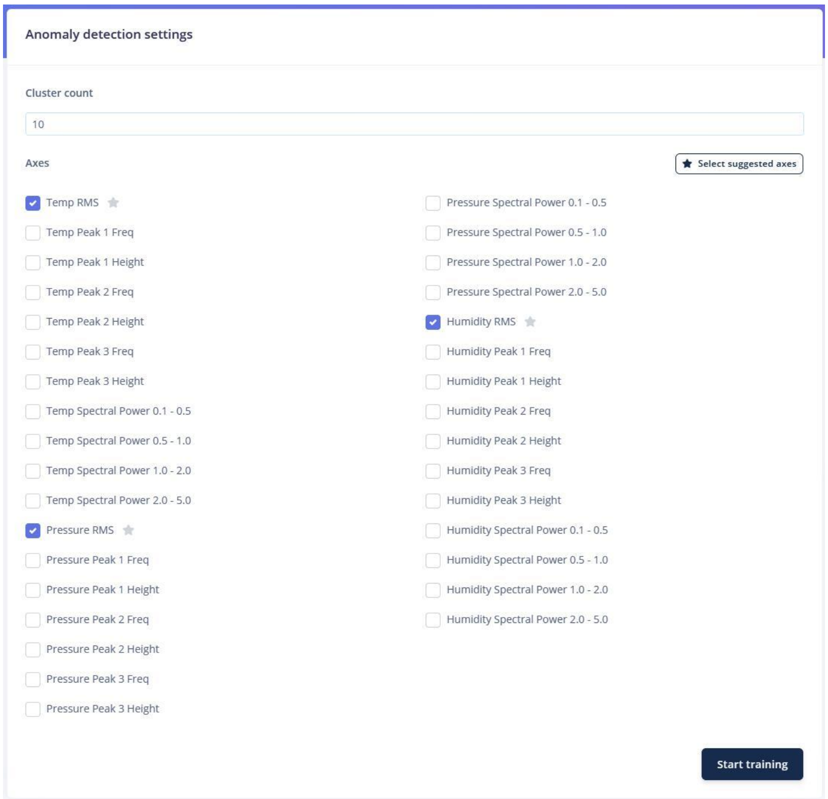


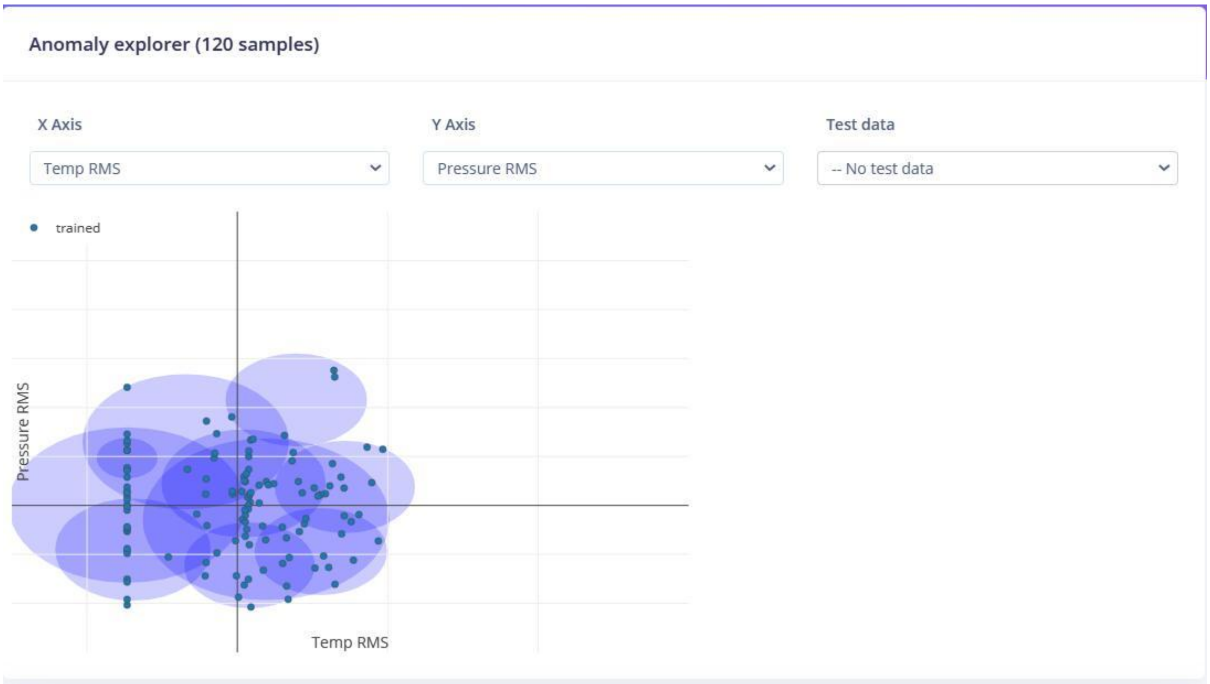
Feature Extraction--Spectral Analysis

The only significant tweak I made was changing the filter from low to high, which made the features more prominent.



Model Training--Network: Anomaly detection





We trainin a network that creates 10 clusters around data that we have seen before and compares incoming data against these clusters. If the distance from a cluster is too large the sample has flagged the sample as an anomaly.

After trial and error, I found that a very low cluster count works the best for anomaly detection, but this is very case-specific and depends on your data.

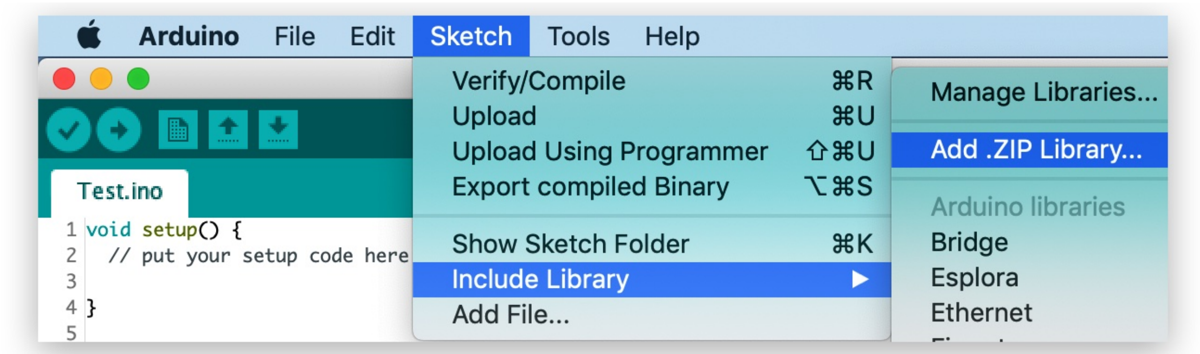
Model Optimization

Model Deployment

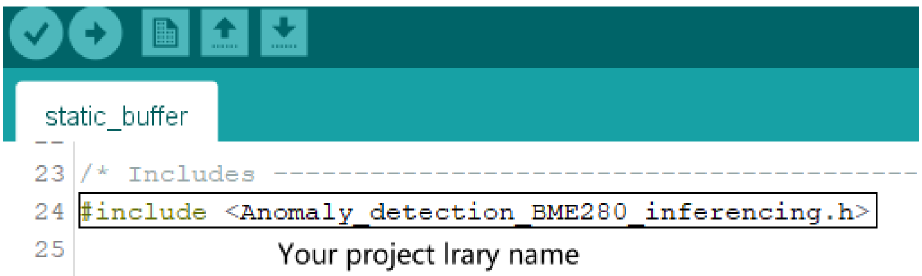
The next step is deployment on the device.

Step 1: After clicking on the Deployment tab, choose Arduino library and download it.

Step 2: Now, the library can be installed to the Arduino IDE. Open the Arduino IDE, click sketch -> Include Library -> Add .ZIP Library, and choose the file that you have just downloaded.



Step 3: Open Examples -> name of your project -> static buffer.



Step 4: Copy the following **Example Code** to replace the original example code:

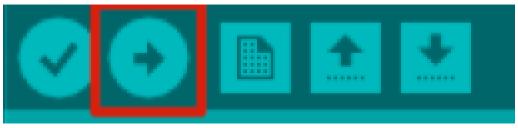
```
#define ANOMALY_THRESHOLD 30

#include "Seeed_BME280.h"
#include <Wire.h>
```

```
#include <Anomaly_detection_BME280_inferencing.h>
#include "TFT_eSPI.h"

TFT_eSPI tft;
BME280 bme280;

static bool debug_nn = false; // Set this to true to see e.g. features generated
from the raw signal

void setup()
{
  Serial.begin(115200);

  tft.begin();
  tft.setRotation(3);

  if(!bme280.init()){
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
  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      Printf function uses vsnprintf and output using Arduino Serial
*
* @param[in]  format     Variable argument list
*/
void ei_printf(const char *format, ...) {
   static char print_buf[1024] = { 0 };

   va_list args;
   va_start(args, format);
   int r = vsnprintf(print_buf, sizeof(print_buf), format, args);
   va_end(args);

   if (r > 0) {
       Serial.write(print_buf);
   }
}

void loop()
{
```

```
  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);

        buffer[ix + 0] = bme280.getTemperature();
        buffer[ix + 1] = bme280.getPressure()/100;
        buffer[ix + 2] = bme280.getHumidity();

        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);

  if (result.anomaly > ANOMALY_THRESHOLD)
  {
    tft.fillScreen(TFT_RED);
    tft.setFreeFont(&FreeSansBoldOblique12pt7b);
    tft.drawString("Anomaly detected", 40, 110);
    delay(1000);
    tft.fillScreen(TFT_WHITE);
  }
```

```
#endif
  Serial.print("Temp: ");
  Serial.print(bme280.getTemperature());
  Serial.println("C");//The unit for  Celsius because original arduino don't
support speical symbols

  //get and print atmospheric pressure data
  Serial.print("Pressure: ");
  Serial.print(bme280.getPressure());
  Serial.println("Pa");

  //get and print humidity data
  Serial.print("Humidity: ");
  Serial.print(bme280.getHumidity());
  Serial.println("%");
}
```

Step 5: Upload the code.



If the upload is successful, the message "Done uploading." will appear in the status bar.

Step 6: Try to simulate an abnormal situation and see whether the Wio Terminal alarms.

Reference

Edge Impulse Public project: https://studio.edgeimpulse.com/public/76507/latest